INTERTEC DATA SYSTEMS®

COMPUSTAR®

USER'S MANUAL

MULTI-USER TERMINAL SYSTEM

NOTICE

All statements, program listings, technical information and recommendations with respect to the product described in this manual, the accuracy or completeness thereof is neither guaranteed nor warranted by Intertec Data Systems Corporation.

Intertec Data Systems Corporation reserves the right to make improvements in the product described in this manual at any time without notice.

CONFIDENTIAL AND PROPRIETARY INFORMATION

Information presented in this manual is furnished for customer reference only and is subject to change.

This document is the property of Intertec Data Systems Corporation, Columbia, South Carolina, and contains confidential and trade secret information. This information may not be transferred from the custody or control of Intertec except as authorized by Intertec and then only by way of loan for limited purposes. It must not be reproduced in whole or in part and must be returned to Intertec upon request and in all events upon completion of the purpose of the loan.

Neither this document nor the information it contains may be used or disclosed to persons not having a need for such use or disclosure consistent with the purpose of the loan without the prior express written consent of Intertec.

COPYRIGHT 1982

USERS MANUAL FOR

INTERTEC'S

COMPUSTAR™

VIDEO PROCESSING UNITS

and

DISK STORAGE SYSTEMS

Document No. 6801030
July, 1982

VPU EMI NOTICE

This equipment complies with the requirements in Part 15 of FCC Rules for a Class A computing device. Operation of this equipment in a residential area may cause unacceptable interference to radio and TV reception requiring the operator to take whatever steps are necessary to correct the interference.

COMPUSTAR
USERS MANUAL REVISION RECORD

REV	DATE	APPROVAL	SHEETS/SECTIONS EFFECTED

TABLE OF CONTENTS

PROPRIETARY STATEMENT

TITLE PAGE i

DISKETTE WARNING NOTICE v

INTRODUCTION AND OVERVIEW ix

COMPUSTAR VPU INTRODUCTION 1-1

Introduction 1-1

VPU Specifications 1-3

VPU Modules 1-6

Keyboard/CPU Module 1-7
CRT Display Module 1-7
Main Power Supply Module 1-9
Chaining Adaptor 1-9
Transition Board 1-10
Disk Drive Modules 1-11

Theory of Operation 1-12

INSTALLATION and OPERATING INSTRUCTIONS 2-1

Unpacking Instructions 2-1

Set Up 2-1

System Diskette 2-1

Reviewing the System Diskette 2-2
Duplicating the Operating Diskette 2-3
Models 30 and 40 2-3
Model 20 2-5
Optional Software 2-6

COMPUSTAR STAND-ALONE SOFTWARE SUMMARY 3-1

CP/M Summary 3-1

CompuStar Model 20 Considerations 3-2

Intertec Utility Summary 3-2

Utility Descriptions 3-3

CONFIGUR.COM 3-3

Vertical Scan Frequency 3-3
Disk Write Verification 3-3
Time Display Enable/Disable 3-3
Key Click Enable/Disable 3-4
Main and AUX Port Operation 3-4

Operating Mode (Main Port Only) . . . 3-4
Baud Rate . . . 3-4
Number of SYNC Characters . . . 3-4
Number of Stop Bits . . . 3-4
Character Length . . . 3-4
Parity . . . 3-4
Handshaking . . . 3-5
SYNC Character Value . . . 3-5
Keypad Reprogramming . . . 3-5

FORMAT.COM . . . 3-5

For Model 30 and 40 VPUs . . . 3-5
For Model 20 VPUs . . . 3-6

HEXDUMP.COM . . . 3-6

64KTEST.COM . . . 3-7

TX.COM . . . 3-7

RX.COM . . . 3-9

TIME . . . 3-9

DATE . . . 3-10

FCOPY . . . 3-10

SECONDARY CHARACTER SET OPTION . . . 3-10

Intertec Secondary Character Sets . . . 3-10
Customer Created Secondary Character Sets . . . 3-10
CSEDIT.COM . . . 3-11
CSDUMP.COM . . . 3-13

MISCELLANEOUS OPERATIONAL INFORMATION . . . 4-1

Using the INP: and OUT: Features of PIP . . . 4-1

Synchronous Communication . . . 4-2

CompuStar VPU Serial Communications DIP Switch . . . 4-3
8251A USART Operation . . . 4-4
Master Reset Feature . . . 4-20
Cursor Control Keys . . . 4-20
Accessing TIME/DATE Data . . . 4-20

Interfacing Information . . . 4-21

RS232-C Serial Interface . . . 4-21

CompuStar Serial Port Pin Assignments . . . 4-21

Bus Adaptor Interface . . . 4-21
Pin Connections for External Bus . . . 4-22
Table of I/O Ports . . . 4-23
Autoload Feature . . . 4-24
Key Click . . . 4-26

Key Repeat 4-26
Type Ahead 4-26
Controlling the Video Display 4-26

Escape Sequences 4-27
Control Codes 4-28
Video Attributes 4-29
Cursor Positioning for Display Control 4-30
Memory Map/Screen Initialization 4-30
Interpreting the ASCII Code Chart 4-33
Control Code Chart 4-34

Wordstar Considerations for CompuStar VPUs 4-34

COMPUSTAR NETWORK OPERATION . . . . . APPENDIX A

Disk System Controller — 96 MB System . . . . . APPENDIX A-1

DSS-10 Disk Storage System . . . . . APPENDIX A-2

CP/M . . . . . APPENDIX B

Introduction to CP/M Features and Facilities . . . . . APPENDIX B-1

Operation of the CP/M Context Editor . . . . . APPENDIX B-2

CP/M 2.0 Users Guide for CP/M 1.4 Owners . . . . . APPENDIX B-3

Operation of the CP/M Debugger . . . . . APPENDIX B-4

Operation of the CP/M Assembler . . . . . APPENDIX B-5

The CP/M 2.0 Interface Guide . . . . . APPENDIX B-6

The CP/M 2.0 System Alteration Guide . . . . . APPENDIX B-7

CUSTOMER INFORMATION . . . . . APPENDIX C

Servicing and Warranty Information . . . . . APPENDIX C-1

General Information for CompuStar Users . . . . . APPENDIX C-2

*** IMPORTANT ***

Do not attempt to write or save programs on your system diskette. It has been write protected' by placing a small adhesive aluminum strip over the notch on the right hand side of the diskette. Such attempts will result in a 'WRITE' or 'BAD SECTOR' error.

Before using your CompuStar VPU for any other purpose, please copy the System Diskette onto a new blank diskette. If you do not have such a diskette, contact your local dealer. He should be able to supply you with one. If you have any questions concerning this procedure, please contact your dealer before proceeding. Failure to do so may result in permanent damage to your System Diskette.

BEFORE APPLYING POWER TO THE MACHINE INSURE THAT NO DISKETTES ARE INSERTED INTO THE MACHINE. NEVER TURN THE MACHINE ON OR OFF WITH DISKETTES INSERTED IN IT. FAILURE TO OBSERVE THIS PRECAUTION WILL MOST DEFINITELY RESULT IN DAMAGE TO THE DISKETTES.

CONGRATULATIONS ON YOUR PURCHASE OF INTERTEC'S COMPUSTAR SYSTEM

Your new CompuStar Video Processing Unit(s) and/or Disk Storage System(s) were manufactured at Intertec's 120,000 square foot plant in Columbia, South Carolina, under stringent quality control procedures to insure trouble-free operation for many years. **If you should encounter difficulties with the use or operation of your VPU, contact the dealer from whom the unit was purchased for instructions regarding the proper servicing techniques.** If service cannot be made available through your dealer, contact Intertec's Customer Services Department at (803) 798-9100.

As with all Intertec products, we would appreciate any comments you may have regarding your evaluation and application of this equipment. For your convenience, we have enclosed a customer comment card at the end of this manual. Please address your comments to:

Product Services Manager
Intertec Data Systems Corporation
2300 Broad River Road
Columbia, South Carolina 29210

Intertec products are distributed worldwide through a network of dealer/OEM vendors and through Intertec's own marketing facilities. Contact us at (803) 798-9100 (TWX — 810-666-2115) regarding your requirement for this and other Intertec products.

INTERTEC DATA SYSTEMS.
Corporate Headquarters: 2300 Broad River Road, Columbia, South Carolina 29210 • 803/798-9100 • TWX: 810-666-2115

Intertec's one hundred and twenty thousand square foot corporate and manufacturing facility in Columbia, South Carolina

WILL THE MICROCOMPUTER YOU BUY TODAY

STILL BE THE BEST MICROCOMPUTER BUY TOMORROW?

Probably the best test in determining how to spend your microcomputer dollar wisely is to consider the overall versatility of your microcomputer purchase over the next three to five years. In the fast-paced, ever-changing world of data communications, new features to increase operator and machine efficiency are introduced into the marketplace daily. We at Intertec are acutely aware of this rapid infusion of new ideas into the small systems business. As a result, we have designed the CompuStar Video Processing Unit in such a manner as to virtually eliminate the possibility of obsolescence.

Many competitive alternatives to the CompuStar available today provide only limited capability for high level programming and system expansion. Indeed, most low-cost microcomputer systems presently available quickly become outdated because of the inability to expand the system. Intertec, however, realizes that increased demands for more efficient utilization of programming makes system expansion capability mandatory. That means a lot. Because the more you use your CompuStar, the more you'll discover its adaptability to virtually any small system requirement. Extensive use of "software-oriented" design concepts instead of conventional "hardware" designs assure you of compatibility with almost any application for which you intend to use the CompuStar.

Once you read our operator's manual and try out some of the features described herein, we are confident that you too will agree with our "top performance — bottom dollar" approach to manufacturing. The CompuStar offers you many more extremely flexible features at a lower cost than any other microcomputer we know of on the market today. The use of newly developed technologies, efficient manufacturing processes and consumer-oriented marketing programs enables us to be the first and only major manufacturer to offer such an incredible breakthrough in the microcomputer marketplace.

Browse through our operator's manual and sit down in front of a CompuStar for a few hours. Then, let us know what you think about our new system. There is a customer comment card enclosed in this manual for your convenience.

Thank you for selecting the CompuStar as your choice for a microcomputer system. We hope you will be selecting it many more times in the future.

INTRODUCTION AND OVERVIEW

This manual documents the CompuStar line of products from Intertec Data Systems. This product line consists of Video Processing Units of varying capabilities and rigid disk storage systems that provide a wide range of storage capabilities for the CompuStar network operations. This line is designed with one consideration in mind; to ensure the versatility that allows the CompuStar concept to meet virtually any small system requirement you will encounter.

This manual can be divided into three topical areas, with each area documenting a different set of information requirements. The first area, "The CompuStar VPU," documents the use of each of the VPUs, Models 10, 20, 30 and 40, in the CompuStar line.

The second area, "CompuStar Networking Operations," consists of appendices which document the integration of the CompuStar VPUs into networks using hard disk drives for mass storage.

The third area, "CP/M," is a reprint of the Digital Research Corporation manuals that document the CP/M* Operating System.

The contents of this manual are distributed with the product(s) which they cover. Therefore you will receive parts of this manual as appropriate for the equipment you have purchased.

The entirety of the manual can be obtained by contacting your Intertec representative or the Customer Services Department at Intertec Corporate Headquarters.

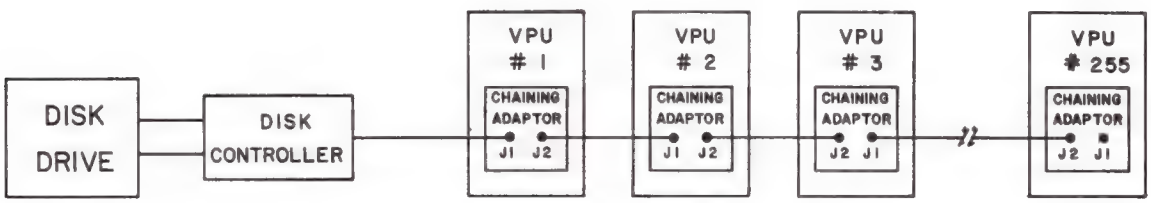
It should be noted that interfacing enhanced CompuStar VPUs with older versions of this product line in a CompuStar network involves many different possibilities. This manual covers the enhanced product line. A separate document will cover the considerations necessitated by the intermixing of enhanced and older VPUs in a network environment. Contact your Intertec representative or the Intertec Customer Services Department if this intermixing is in your plans for CompuStar use.

*CPM is a registered trademark of Digital Research

The CompuStar network concept is an extremely versatile one. When configured as a multi-user system, the CompuStar network consists of a number of video processing units (VPUs), which employ their own internal microprocessors and dynamic RAM (Random Access Memory), a controller and hard disk storage. The terminals are all interfaced with the hard disk storage facility through a controller and "share" the resources of the hard disk unit.

The CompuStar network allows true multi-user, multi-processing capability. Each VPU is set to a unique ID number by the user. This permits the sharing of a common data base while at the same time maintaining a restricted data base for each VPU. Additionally, each VPU can contain up to 1.5 MB of on-board floppy disk storage. These capabilities make the CompuStar concept particularly well-suited for distributed data processing and small business environments. The figure below shows a typical CompuStar system.

For the user who does not currently require a multi-user capability, the CompuStar Model 30 and 40 VPUs represent an ideal choice. They operate as stand-alone VPUs and preserve the ability to later expand to a multi-user network.



CompuStar System

COMPUSTAR VIDEO PROCESSING UNITS INTRODUCTION

COMPUSTAR VPU INTRODUCTION

INTRODUCTION

This material is intended to document the various CompuStar VPU models primarily from the viewpoint of their stand-alone operation.

* CompuStar Model 10 — Intended for use in a network environment only. Contains no floppy disk capability.
* CompuStar Model 20 — May be used stand-alone but more appropriate for a network environment. Contains one floppy disk drive that provides 350K bytes of formatted on-line floppy disk storage.
* CompuStar Model 30 — Contains two floppy disk drives that provide up to 700K bytes of formatted floppy disk storage.
* CompuStar Model 40 — Contains two floppy disk drives that provide up to 1.5 megabytes of formatted on-line floppy disk storage.

It should be noted that the CompuStar Model 10 is intended for use only in a multi-user CompuStar network. As such, some considerations will be covered in these sections that will not apply to the Model 10, since these sections will document the use of the CompuStar VPUs as stand alone, single user microcomputers. The specific information needed to operate these VPUs in a network environment will be presented in later sections where each use of the network operation options is documented. The basic use of the VPU is generally the same from the user's viewpoint in either a stand alone or network environment.

The CompuStar VPUs represent the latest technological advances in the microprocessor industry. The universal adaptability of the CompuStar CP/M Disk Operating System satisfies the general purpose requirement for a low cost, high performance microcomputer system.

From the standpoint of human engineering, the CompuStar Video Processing Unit (VPU) has been designed to minimize operator fatigue through the use of a typewriter-oriented keyboard and a remarkably clear display. The CompuStar VPU displays a total of 1,920 characters arranged in 24 lines with 80 characters per line. The video display characters can be varied between a primary and secondary character set. Blinking, half, intensity, underlining, and reverse video are user selectable display options. The video display is crisp and sharp due to Intertec's own specially designed video driver circuitry. And, the high quality, non-glare etched CRT face plate featured on every CompuStar VPU assures ease of viewing and uniformity of brightness throughout the entire screen.

The CompuStar VPU's unique internal design assures users of exceptional performance for just a fraction of what they would expect to pay for such "big system" capabilities. The CompuStar VPU utilizes a single board "microprocessor" design which combines all processor, RAM, ROM, disk controller, and communications electronics on the same printed circuit board. This type of design engineering enables the CompuStar VPU to deliver superior, competitive performance.

Standard features of the CompuStar VPU include: two mini-floppy disk drives with up to 1.5 megabytes **formatted** disk storage, (Note: The Model 10 has no floppy disk storage and the Model 20 has one floppy disk drive) 64K of dynamic RAM memory, a recognized CP/M Disk Operating System featuring its own text editor, an assembler for assembly language programming, a program debugger and a disk formatter. Also standard are dual universal RS232 communication ports for serial data transmission between a host computer network via modem or an auxiliary serial printer. A number of transmission rates up to 9600 baud are available and selectable under program control.

Other standard features of the CompuStar VPU include: special operator convenience keys, dual "restart" keys to insure simplified user operation, a full numeric keypad complement (whose values can be user reassigned by software when used in a stand alone mode), and a high quality typewriter compatible keyboard. Additionally, a TIME/DATE clock is incorporated for time/date display and is user accessible.

For reliability, the CompuStar has been designed around six (6) basic modules packaged in an aesthetically pleasing desk-top unit. These major components are: the Keyboard/CPU module, the power supply module, the CRT assembly, the transition board, the chaining adaptor, and the disk drives themselves (except the Model 10). Failure of any component within the terminal may be corrected by simply replacing only the defective module. Individual modules are fastened to the chassis in a manner that facilitates easy removal and installation.

Terminal down-time can be greatly minimized by simply "swapping-out" one of the modules and having component level repair performed at one of Intertec's Service Centers. Spare modules may be purchased from an Intertec marketing office to support those customers who maintain their own "in-house" repair facilities.

All CompuStar VPU cover assemblies are exclusively manufactured "in-house" by Intertec. A high-impact structural foam material is covered with a special "felt-like" paint to enhance the overall appearance. Since the cover assembly is injected-molded, there is virtually no possibility of cracks and disfigurations in the cover itself. By manufacturing and finishing the cover assemblies in-house, Intertec is able to specify only high quality material on the external and internal cover components of your CompuStar VPU to insure unparalleled durability over the years to come.

A wide variety of programming tools and options are either planned or available for the CompuStar VPU. Software development tools available from Intertec include Basic (standard) and Fortran (optional) programming languages. A wide variety of applications packages (general ledger, accounts receivable, payroll, inventory, word processing, etc.) are available to operate under CP/M Disk Operating System from leading software vendors in the industry.

Disk storage capability is expandable by interfacing the CompuStar VPU to a rigid disk which makes up to 384 megabytes of on-line storage available. This interface is easily accomplished by assigning a unique VPU ID number to each VPU, initializing the appropriate hard disk device, and chaining the VPUs together through the chaining adaptor in each VPU. This allows up to 255 users to share system network resources and still retain individually the full processing power of each VPU.

The high performance ratio of the CompuStar has rarely been equalled in this industry. By employing innovative design techniques, the CompuStar is not only able to offer a competitive price advantage but boasts many features found only in systems costing three to five times as much. The CompuStar VPU twin Z80A microprocessors insure extremely fast program execution even when faced with the most difficult programming tasks. Additionally, each unit must pass a grueling 72 hour burn-in before it is shipped to the customer. By combining advanced microprocessor technology with in-house manufacturing capability and stringent quality control requirements, your CompuStar VPU should provide unparalleled reliability in any application into which it is placed.

VPU SPECIFICATIONS

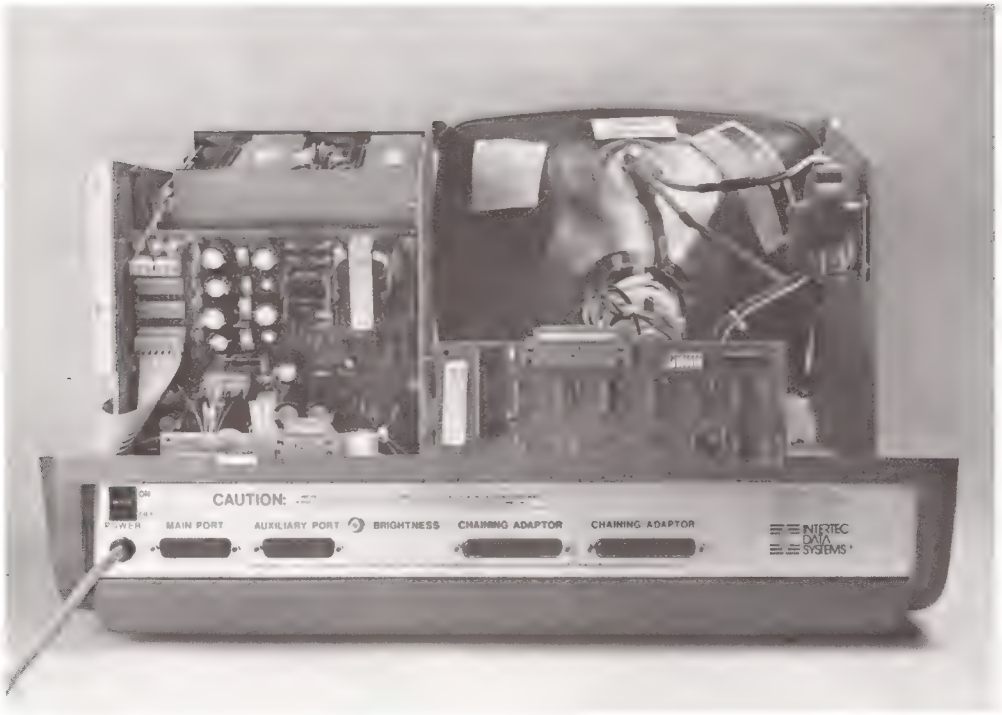
FEATURE	DESCRIPTION
CPU:	
Microprocessors	Twin Z80A's with 4MHZ Clock Frequency. One Z80A (the host processor) performs all processor and screen related functions. The second Z80A is "down-loaded" by the host to execute disk I/O. Not applicable to Model 10.
Word Size	8 bits
Execution Time	1.0 microsecond register to register
Machine Instructions	158
Interrupt Mode	All interrupts are vectored
FLOPPY DISK:	
Storage Capacity (Formatted)	Model 10 — none, Model 20 — 350K, Model 30 — 700K, Model 40 — 1.5 MB
Data Transfer Rate	250K bits/second
Average Access Time	250 milliseconds. 6 milliseconds track-to-track.
Media	5¼-inch mini-floppy disk
Disk Rotation	300 RPM
INTERNAL MEMORY:	
Dynamic RAM	64K bytes dynamic RAM
Static RAM	2048 bytes of static RAM is provided in addition to the main processor RAM. 1K x 8 of this RAM storage is used as a disk buffer. The remaining RAM is used for attribute storage.
Firmware	2K x 8 bytes standard. Allows "bootstrapping" of system at power-on and contains micro codes for disk control.
Time/Date Clock	Provides continuous time display. Maintains time and date information during power-off and compensates for variances in month/year lengths.
CRT:	
Display Size	12 inch, specially focused, P4 phosphor, non-glare screen.
Display Format	24 lines x 80 characters per line

FEATURE	DESCRIPTION
Character Font	5 x 7 character matrix (with descenders) on a 7 x 10 character field. All displayed characters are derived from character sets stored on interchangeable EPROMS.
Display Presentation	Light characters on a dark background. Blinking, half-intensity, underlining, reverse video attributes standard; optional on-line secondary Character/Graphic set.
Bandwidth	20 MHZ
Cursor	Reversed image (block cursor)
COMMUNICATIONS:	
Screen Data Transfer	Memory-mapped at 38 kilobaud.
Auxiliary Interface	Simplified RS-232 asynchronous. Baud rates are software selectable from 50 to 9600 baud.
Main Interface	Universal RS-232 asynchronous/synchronous. Baud rates software selectable from 50 to 9600 baud.
Transparent Mode	Enables display of all incoming and outgoing control codes.
Parity	Choice of even, odd, none
Transmission Mode	Half or Full Duplex. One, one and one-half, or two stop bits.
Addressable Cursor	Direct positioning by either discrete or absolute addressing.
SYSTEM UTILITIES:	
Disk Operating System	CP/M 2.2
DOS Software	An 8080 disk assembler, debugger, text editor and file handling utilities.
BASIC	Sequential and random disk access. Full string manipulation, interpreter.
OPTIONAL SOFTWARE:	
LANGUAGES	FORTRAN; ANSI standard with relocatable, random and sequential disk access. Additionally, any user furnished CP/M compatible software package that can reside in 52K of memory.

FEATURE	DESCRIPTION
Application Packages	Extensive software development tools are available including software for the following applications: Payroll, Accounts Receivable, Accounts Payable, Inventory Control, General Ledger and Word Processing. Contact an Intertec Sales Office for complete details.
KEYBOARD:	
Alphanumeric Character Set	Generates all 128 upper and lower case ASCII characters.
Special Features	N-key rollover, type ahead, and key repeat
Numeric Pad	0-9, decimal point, comma, minus and four cursor control keys. Reprogrammable to other values for individual applications.
Cursor Control	Up, down, forward, backward and home.
INTERNAL CONSTRUCTION:	
Cabinetry	Structural foam
Component Layout	Five board modular design. All processor related functions, RAM, controllers and keyboard are on a single printed circuit board. All video, chaining, and power related circuits on separate boards.
ENVIRONMENT:	
Weight	Approximately 45 pounds
Physical Dimensions	14-5/8'' (H) x 21-3/8'' (W) x 23-1/8'' (D)
Environment	Operating 0° to 50° C Storage: 0° to 85° C; 10 to 95% relative humidity — non condensing.
Power Requirements	115 (±10%) VAC/60 HZ, 1 AMP (optional 230 VAC/50HZ model available)

VPU MODULES

Perhaps the most remarkable feature of the CompuStar VPU is its modular construction using only six major subassemblies which are clearly defined in their respective functions so as to facilitate ease of construction and repair. These six subassemblies are shown and described in the following. Additionally, numerous optional software packages are available for use with your CompuStar VPU. If you would like additional information on these packages, please contact your local Intertec representative.



Keyboard/CPU Module

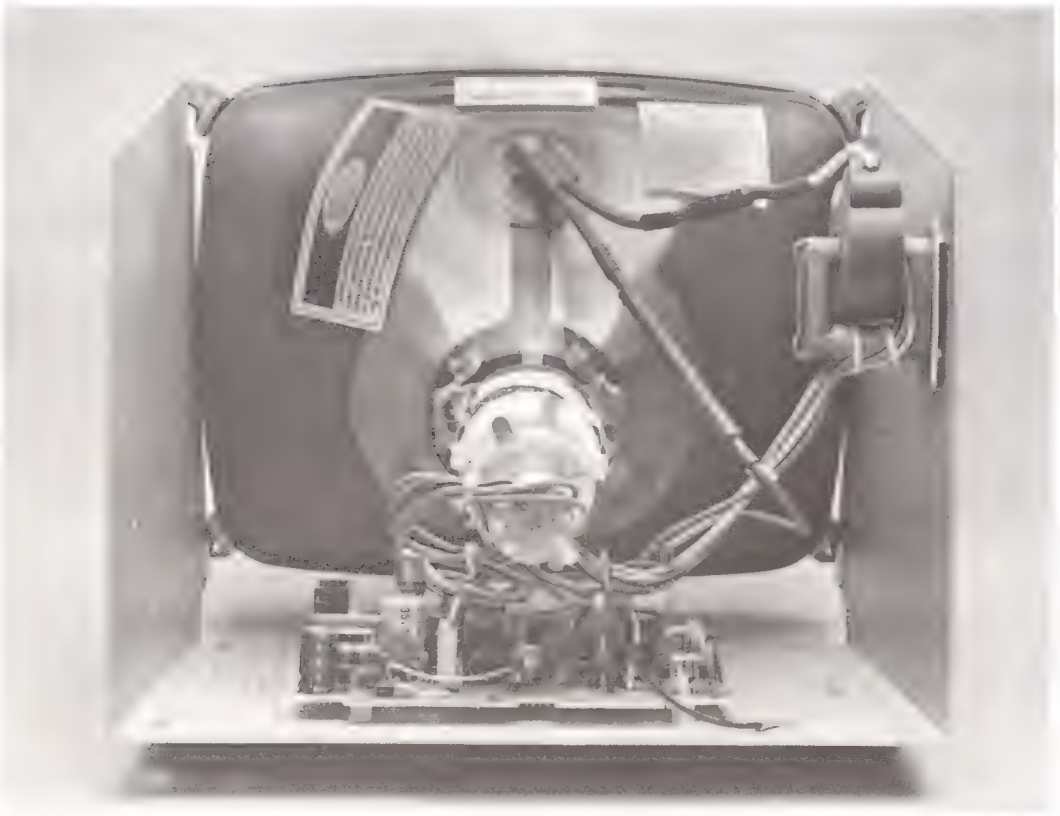
The control section of the CompuStar VPU is based upon the widely acclaimed Z80A microprocessor. The result is far fewer components and the ability to perform a number of functions not possible with any other approach. The Keyboard/CPU module contains the CompuStar twin Z80A microprocessors. One Z80A (the host processor) performs all processor and screen related functions while the second Z80A can be "downloaded" to execute disk I/O handling routines. The result is extremely fast execution time for programs. The Model 10, by virtue of not having floppy disk drives, has one Z80A.

In addition to containing the CompuStar VPU microprocessor circuitry, the Keyboard/CPU module contains 64K of dynamic RAM. Also found on this module is: the character and keyboard encoder circuitry, the "bootstrap" EPROM, the disk controller and all communications electronics. Power is supplied to this module via a single 7 pin ribbon cable connected to the CompuStar main power supply module. Connection of this module to the disk drive modules is via a separate ribbon cable. Separate connectors also exist for the CRT display signals and serial I/O ports.

CRT Display Module

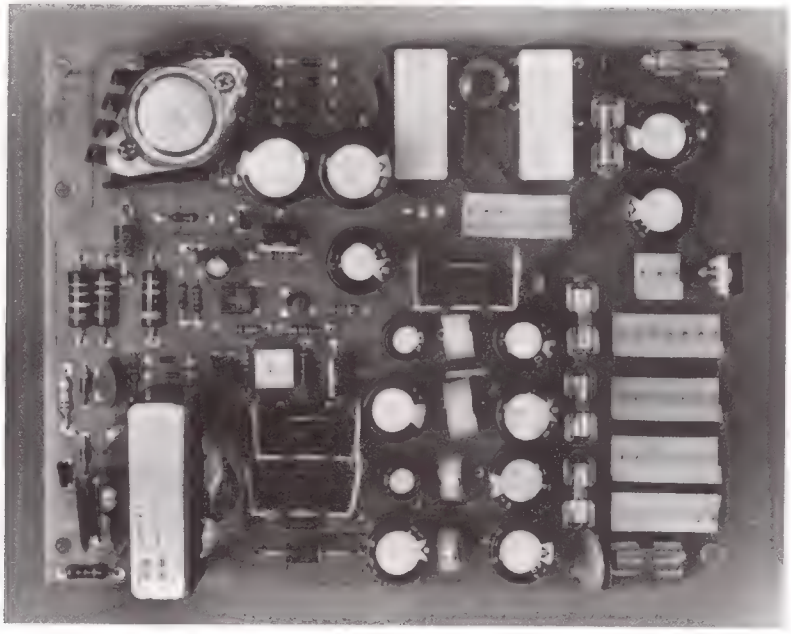
The CRT Display Module consists of a 12 inch, high resolution, cathode ray tube mounted in a rigid aluminum chassis. The faceplate of the CRT is etched in order to reduce glare on the surface of the screen and provide uniform brightness throughout the entire screen area. The CRT display presentation is arranged in 24 lines of 80 characters per line for a total display capacity of 1,920 characters.

The CRT video driver circuitry is mounted in the base of the CRT chassis to facilitate ease of removal and subsequent repair. In this manner, either the CRT itself or the video circuitry can be easily exchanged without disrupting any of the other major modules within the terminal.



Main Power Supply Module

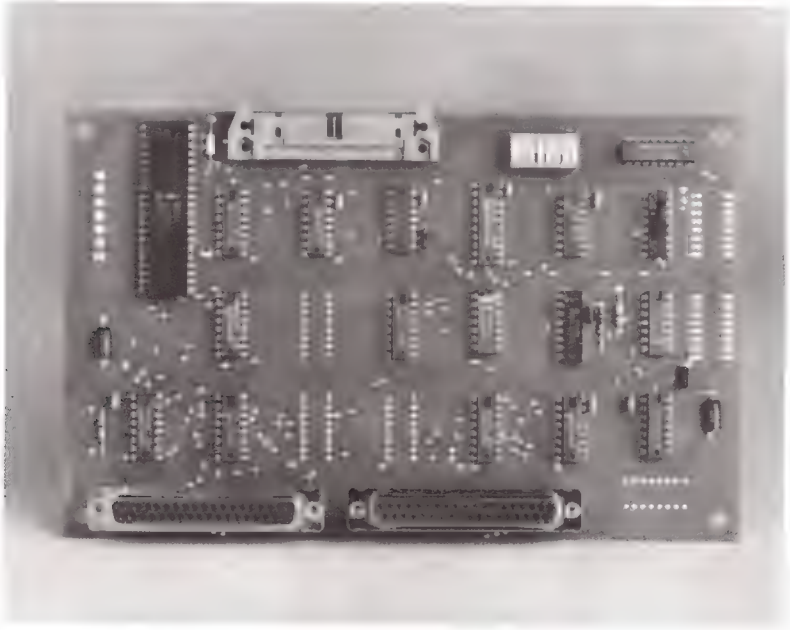
The CompuStar power supply is of a "solid-state, switching" design and employs a voltage regulator to provide many years of trouble-free service. This design reduces heat dissipation and allows for efficient cooling of the entire terminal with a specially designed whisper fan to reduce environment noise. The entire power supply can be easily removed by unscrewing the screws holding it to the disk drive back plate. This module supplies the five voltages required to power the Keyboard/CPU module, the Video Module, and disk drive.



Chaining Adaptor (Unique to CompuStar)

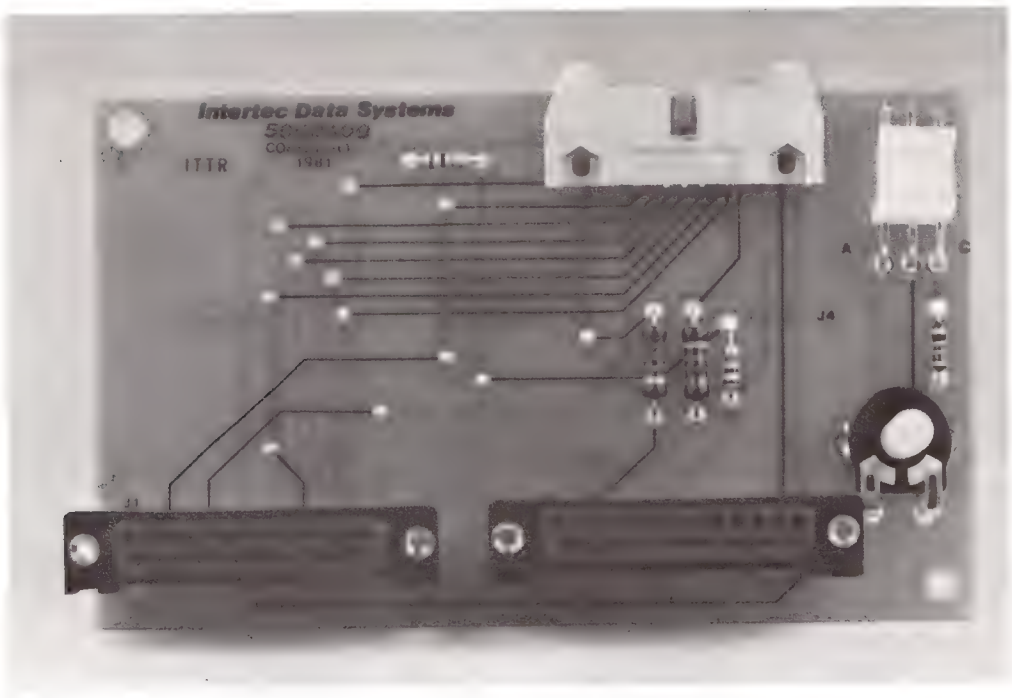
Each CompuStar VPU contains a chaining adaptor which allows up to 255 VPUs to operate in one network and share in the resources of a hard disk drive.

A dipswitch on the chaining adaptor is set by the user to designate a VPU ID for each VPU in a network.



Transition Board

This board contains the RS-232 serial I/O connectors and video brightness control. It connects to the video module and the keyboard/CPU module.

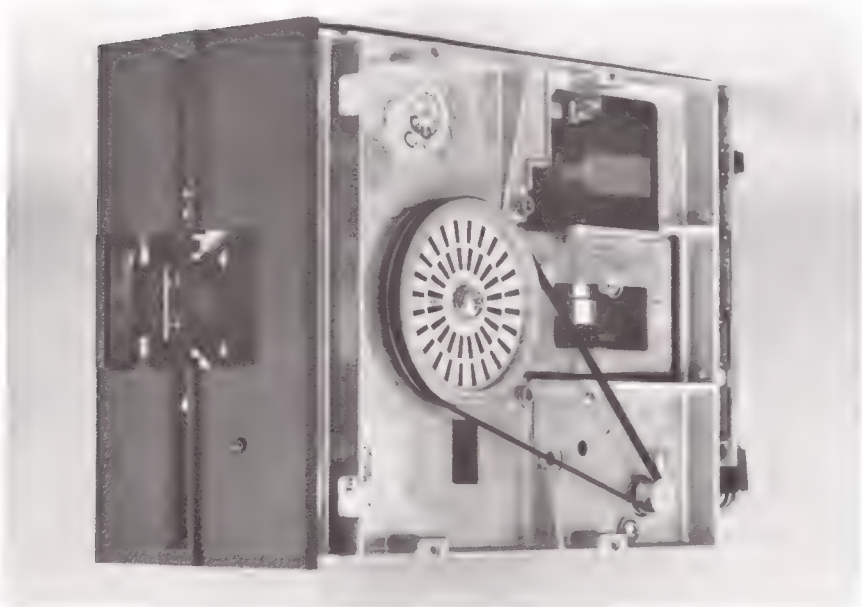


Disk Drive Modules

The CompuStar VPUs contain varying amounts of floppy disk storage, except the Model 10, which has none (see chart below). The drives used are double sided, double density, and 48 or 96 TPI. This results in a range of floppy disk storage up to 1.5 megabytes for the CompuStar Model 40. The CompuStar Model 20 has only one floppy disk drive.

These drives are mounted conveniently just to the right of the CRT display module on a rugged aluminum mounting bracket so that they are flush mounted with the front "bezel" of the unit. Power to these drives is derived from the Power Supply Module located just behind the drive assemblies themselves. Data to and from these drives is routed via a single 34 pin ribbon cable connecting the drives to the Keyboard/CPU module.

Front View of CompuStar Disk Drive Module



Side View of CompuStar Disk Drive Module

THEORY OF OPERATION

The CompuStar VPU, Models 20, 30, and 40 contain two Z80 microprocessors. (Model 10 has only one Z80A because there are no flexible disk drives.) uP1 is the main processor which executes all user programs from the 64K RAM main memory, while transparently managing the CRT Display processes. All user I/O is also connected to uP1. This I/O includes the Serial Ports, Interface Controller, Keyboard Encoder, Time/Date Clock, and the External Bus. uP2 performs all floppy disk control functions from instructions contained in the 2K Bootloader EPROM. Part of this same EPROM contains the Cold Bootloader for uP1, and is executed when a System Reset is performed. The Floppy Disk Control section also contains a 1K x 8 RAM buffer used for temporary storage of disk read/write data. This buffer can be accessed by either uP1 or uP2, therefore, a protocol exists to prevent microprocessor contention for this buffer.

The 64 kilobyte main memory consists of thirty-two 16K x 1 bit dynamic RAMS. These are divided in four banks (0-3) with each bank containing 16 kilobytes of storage. The RAS-CAS timing sequence necessary for memory access is created by the memory timing generator.

The CRT-VIDEO CONTROLLER circuitry is divided into three main areas: The CRT controller which generates all the timing signals for data display; the character generator circuitry which produces the character font; and the attribute generation circuitry which provides the special video capabilities of blinking, underlining, half-intensity, and reverse video in addition to normal video display.

The capability exists to install an alternate character set EPROM as an option. This would allow the CRT controller to access either character set during normal operation.

The CRT controller generates all the timing necessary to display 24 rows of characters with 80 characters per row. Thus the screen can display a total of 1,920 characters. These characters are stored in the CRT refresh buffer which is the upper 2,048 bytes (2K) of main memory.

Because the CRT buffer is not a separate buffer and the processor must also use the same bus to access memory, this bus must be time-shared between the two. This is accomplished by the CRT controller performing a direct memory access (DMA) cycle which is done at the last scan line of each character row. Each character row is divided into ten scan lines, therefore, during the last scan line time, the controller takes control of the processor bus by generating a bus request. After acquiring the bus, it reads 80 characters from the CRT buffer and loads them into the 80 x 8 shift register. This data is then recirculated in the buffer for the next nine scan lines to produce one row of video characters. Therefore, there are twenty-four DMA cycles performed per vertical frame.

There are also twenty-five interrupts generated — one for each row scan and one extra during vertical blanking. During the first twenty-four, the processor sets or resets the video blanking depending on whether that row is displayed or not. During the vertical blanking interrupt, the address registers in the CRT controller are initialized to the correct top-of-page address and the cursor register is also updated.

The Interface Controller is basically three 8 bit I/O ports (8255). Through this device, the processor can obtain status bits from other devices and react to the status by setting/resetting individual bits in the 8255.

The Keyboard Encoder scans the keyboard for a key depression, determines its position, and generates the correct ASCII code for the key. The processor is flagged by the 'Data Ready' signal via the Interface Controller. The character is then input by the processor.

The Time/Date clock is accessed directly by uP1 through an I/O address. The clock has a battery

power supply and will maintain the correct time and date when the external power is removed. The clock is also available as a real-time clock for the user's access.

There are also two RS232C serial interface ports. The main port is capable of synchronous or asynchronous operation. The aux port is a simplified port used for asynchronous operation only. The baud rates are variable from 50 baud to 9600 baud. The mode of operation of the main port and the baud rate of both ports are set up by the operating system and can be changed by using the "CONFIGUR" program.

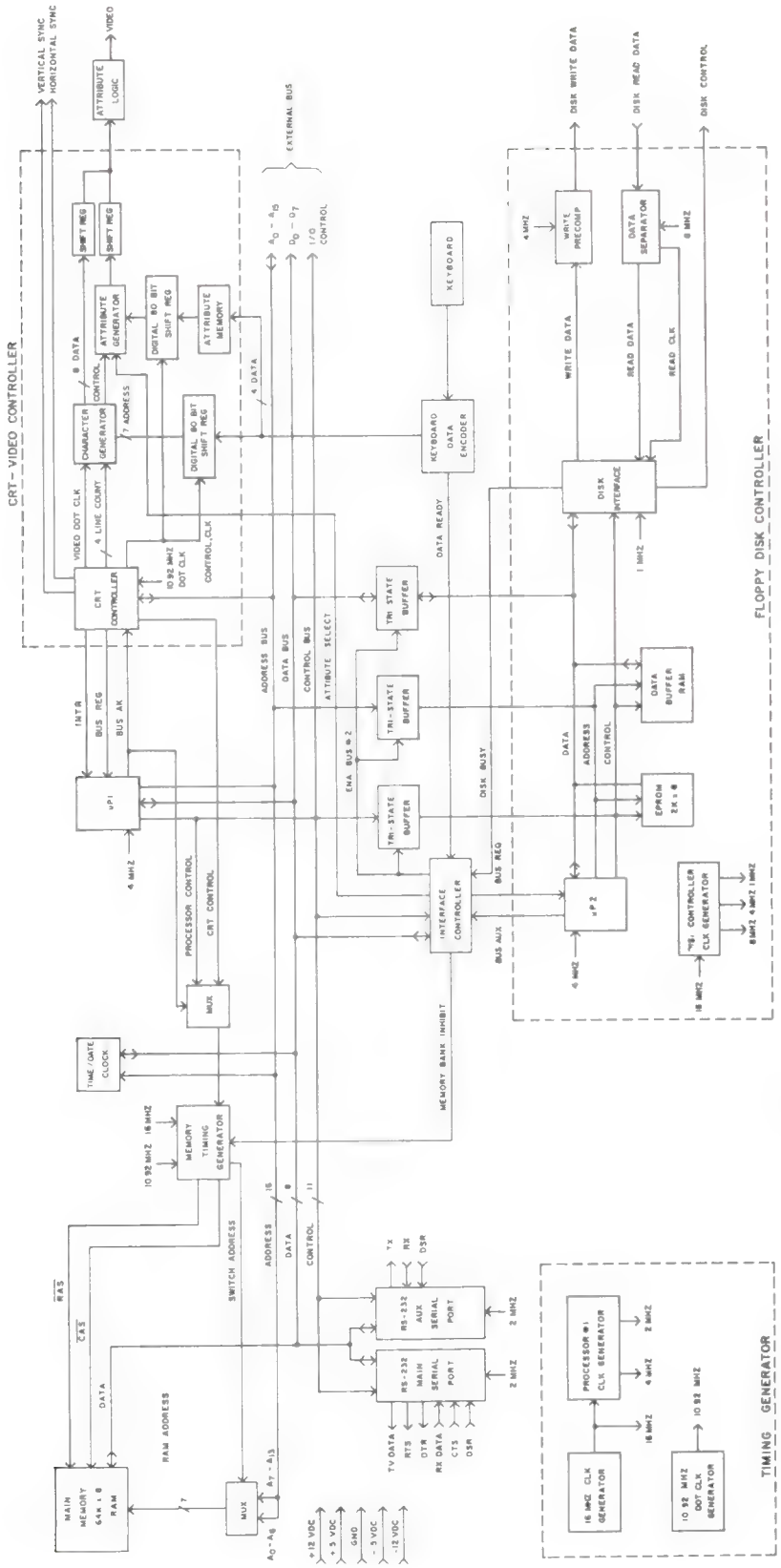
As previously mentioned, uP1 has the capability of communicating with the RAM and ROM in the FLOPPY DISK CONTROLLER. Because the amount of main memory used is the maximum that the processor addressing can support, different 16K banks of main memory must be switched off line when communicating with the disk RAM or EPROM. In these cases Bank 0 (0000H-3FFFH) is switched out when communicating with the EPROM, and Bank 2 (8000H-BFFFH) when communicating with the RAM.

The FLOPPY DISK CONTROLLER performs all disk related I/O functions upon command from the main processor. These commands are:

* Restore to track 0
* Read sector
* Write sector
* Write sector with verify
* Format

When the parameters associated with drive, side, track, and sector numbers are loaded, a status word is set at a specified location in the disk RAM. When uP2 receives this status, it sets the 'disk busy' status bit and performs the indicated function. Upon completion, it resets the 'busy' bit thus allowing the main processor (uP1) to retrieve data and status from the RAM.

This design reinforces the flexibility of the CompuStar concept; efficient VPUs of varying capabilities able to share hard disk storage resources while still maintaining the benefits of independent operation.



SUPERBRAIN II KEYBOARD / CPU II MODULE BLOCK DIAGRAM

INSTALLATION & OPERATING INSTRUCTIONS

INSTALLATION AND OPERATING INSTRUCTIONS

CompuStar VPU Models 20, 30, and 40 may be operated as stand-alone computers using the CP/M disk based operating system. Model 10 VPUs have no built in floppy disk storage and must be used in a CompuStar network environment.

The following information primarily applies to Model 20, 30, and 40 VPUs that are used as stand-alone computers. Exceptions to this are noted where they occur.

UNPACKING INSTRUCTIONS

Be sure to use extreme care when unpacking your CompuStar VPU. The unit should be unpacked with the arrows on the outside of the shipping container facing up.

The MASTER SYSTEM DISKETTE is located inside the front cover of this manual. Be careful not to discard or misplace this diskette as it will be vital for the later operation of the equipment. If you ordered additional, optional software with your computer, it will be shipped under separate cover.

Now that you have located your system diskette you can proceed to remove the terminal. If you should experience any difficulties, rotate the carton on its side. With the terminal in this position, you should now be able to pull outward on the terminal and separate it from the box. Once the terminal is out of the carton, place it on a table and remove the protective plastic bag which should be surrounding the terminal. DO NOT DISCARD THE SHIPPING CARTON SINCE IT COULD POSSIBLY BE USED FOR RESHIPPING AT A LATER DATE.

SET UP

The first step in this procedure is to verify that your CompuStar VPU is wired for a line voltage that is available in your area. This can be ascertained by checking the serial tag located at the right rear of the terminal. This tag should indicate that your unit is set up for either 110 or for a 220 VAC operation. DO NOT ATTEMPT TO CONNECT THE COMPUSTAR VPU TO YOUR LOCAL POWER OUTLET UNLESS THE VOLTAGE AT YOUR OUTLET IS IDENTICAL TO THE ONE SPECIFIED ON THE BACK OF YOUR TERMINAL. If the voltages differ, contact your dealer at once and do not proceed to connect the CompuStar VPU to the power outlet.

Before connecting the CompuStar VPU to the wall outlet, be sure that the power switch located at the left rear corner is turned OFF, and the cardboard insert(s) in the flexible disk drive(s) is/are removed. You may now proceed to connect your computer system to the wall outlet. After completing this connection, turn the power switch to the ON position. At this time, you should hear a faint "whirring" sound coming from the fan in the computer. After approximately 60 seconds the message INSERT DISKETTE INTO DRIVE A will appear on the screen. If this message does not appear on the screen after approximately 60 seconds, simultaneously depress the two RED keys located on either side of the alphanumeric keyboard. These are the master system reset keys and should reinitialize the computer system, thereby displaying the 'INSERT' message on the screen. If, after several attempts at resetting the equipment you are unable to get this message to appear on the screen, turn the unit off for approximately 3 to 5 minutes and then reapply power to the unit. If you are still unable to get the appropriate message to appear on the screen, contact your Intertec representative.

SYSTEM DISKETTE

Now that you have power applied to the machine and the INSERT DISKETTE message has been displayed in the upper left corner, you are ready to proceed with loading the computer's operating system. This is accomplished by locating the small 5¼" diskette that was packed with this manual. Once you have located this diskette, you will notice that a small adhesive strip has

been placed over the notch on the right side of the diskette. This aluminum strip is used to "WRITE PROTECT" the diskette. Therefore, you may only read programs from this diskette. If you wish to write or save programs on the system diskette, it will be necessary to remove the small adhesive aluminum strip from the diskette. This is **NOT RECOMMENDED** as it will subject your diskette to accidental errors that may be caused by you while you are getting familiar with the operating system.

You are now ready to proceed with inserting the system diskette into the machine. When facing the front of the machine, you will notice that there are two small openings on the right side of the machine (except for the Model 10 which has none, and the Model 20 which has only the leftmost opening). The leftmost opening is designated as drive A. The rightmost opening is designated as drive B. This distinction is important since the disk operating system can only be loaded from drive A.

Open the disk drive door on drive A (the leftmost drive). The drive can be opened by applying a very slight pressure outward on the small flat door located in the center of the opening. Once the drive door has been opened, insert the Operating System Diskette. The front of the diskette should contain a small white sticker located in the upper left corner of the diskette. This diskette should contain a message indicating that it is the CompuStar Stand-Alone DOS Diskette with CP/M Version 2.2. Be careful to insure that (1) the small aluminum write protect strip is oriented towards the top edge of the diskette and that (2) the label located in the upper left corner of the operating system diskette is facing AWAY from the screen towards the right side of the terminal. Once you have oriented the diskette in this fashion, insert it into the terminal.

It is **EXTREMELY** important that the diskette be properly oriented before inserting it into the machine since improper orientation will not allow the operating system to properly load. Once the diskette has been placed in the machine, be sure that it has been inserted all the way by applying a gentle pressure on the rear edge of the diskette. Once you are certain that the diskette is fully inserted, close the disk drive door. This can be accomplished by applying a slight pressure on the door, pulling it back into the direction from which it was originally opened. Once you have closed the door, you will notice a small "swishing" sound. This sound is normal and indicates that the computer is now attempting to load the operating system. Some drives are quieter than others and therefore this noise may not be audible.

After closing the door the following message should appear in the upper left corner of the screen:

COMPUSTAR DOS VER x.x, FOR CP/M 2.2

A>

If this message does not appear on the screen, try depressing the two RED keys located on either side of the keyboard. This should reset the terminal and thereby attempt to reload the operating system. If after several seconds, the message does not appear on the screen, try depressing the RED keys several more times. If repeated depressions of the RED keys do not bring up the indicated message, then open the door on the disk drive A and remove the system diskette and check to see if it was properly inserted. If you are unsure as to the proper orientation of the diskette, please contact the representative from whom you originally purchased your equipment.

Reviewing the System Diskette

After you have successfully loaded the System Diskette and Disk Operating System (DOS), the CompuStar VPU is ready to accept your disk operating system commands. At this time we will review several of the commands in the operating system. However, it is recommended that you refer to the appropriate section in this manual for a detailed description of all such commands. (Introduction to CP/M Features and Facilities). The most used system command is the DIR

command. This command directs the operating system to display the directory of all programs contained on the system diskette. You may enter this command by simply typing the letters **DIR** on the keyboard.

After you have typed these letters, it is necessary to depress the **RETURN** key. Depressing this key instructs the computer to process the line of data that you have just typed. After you depress the RETURN key the computer should respond by displaying all of the programs on the system diskette. These programs will appear in a form somewhat similar to the following:

A:ED.COM
A:DDT.COM
A:ASM.COM
A:LOAD.COM
A:DUMP.COM

To obtain a better understanding of just what this information means, let's take a look at the first line:

A:ED.COM

The first letter on this line is the letter A. This tells you that the information following this letter is located on drive A. The colon serves as a separator between the drive designator ("A") and the file NAME and file TYPE. The file NAME is, in this case, "ED" and the file TYPE is "COM." This line tells the operator that a program called ED (the disk operating system text editor) is located on the "A" drive and is a COM type of file. A more detailed treatment of this information can be found in the CP/M sections of this manual.

Duplicating the Operating Diskette

Now that you have successfully loaded the Disk Operating System on Drive A, it is important to duplicate this diskette. This is necessary in order to preserve the original copy of the diskette and guard against any possible damage to the original media. To generate a copy of the operating system you will first need a new blank diskette. We recommend an Intertec diskette for this purpose. If you do not have any blank diskettes of similar quality, please contact the representative from whom you purchased your equipment. The representative should be able to supply you with an ample quantity of these diskettes.

MODELS 30 AND 40

For CompuStar Model 30 and 40 VPUs insert the blank diskette into drive B. Follow the procedures outlined in the previous paragraphs regarding the insertion of the operating system diskette. The only difference is that you will be inserting the new blank diskette into drive B. Be sure and leave the system diskette installed in drive A.

Once you have installed the new blank diskette in drive B, you are ready to "FORMAT" the new diskette. It is necessary to format all previously unused diskettes before attempting to transfer data to them. This is necessary because all information is stored on diskettes in what is known as a SOFT SECTORED FORMAT which necessitates the writing of certain information on the disks before user programs or data can be stored on them.

To format the diskette in drive B, enter the command **FORMAT** and depress the **RETURN** key. The operating system will respond by asking you to select one of the following:

* 3 — For formatting CompuStar Model 30 diskettes.
* 4 — For formatting CompuStar Model 40 diskettes.

CAUTION: CompuStar Model 20 and 30 diskettes cannot be formatted on a Model 40 VPU and vice-versa.

Once the appropriate option is selected, the operating system will prompt the user to insert a blank diskette into drive B in case that has not already been done. Next, the user should depress the **F** key to begin formatting.

When a diskette is being formatted, the read/write heads position to track 0 and sequentially write each track. The screen displays the current track numbers. The track value displayed will range from:

* 0-69 for the CompuStar Model 30.
* 0-159 for the CompuStar Model 40.

After the disk has been completely formatted, the operating system will respond by asking you whether to "REBOOT" the operating system or whether you wish to format another disk. If you wish to format another disk, remove the newly formatted disk from drive B and insert a new blank diskette into drive B. You may now proceed to format this new diskette by once again entering the letter **F**. If you do not wish to format any more diskettes, simply enter a **RETURN**.

The Operating System should now reload and once again be ready to accept new commands.

Since the intent of this procedure was to copy the original disk operating system we are now ready to begin that procedure. This can be accomplished by entering the following command on the keyboard:

A >**PIP B:=*.*[V]** <cr>

The system will now begin to copy and verify all of the programs on drive A over to drive B. As each program is copied, its name will be displayed on the screen. This procedure takes approximately 5 to 10 minutes. When the procedure completes, the control of the operating system will be returned to the user.

Now that you have completed copying the programs from the A drive to the B drive it is necessary to copy the disk operating system itself (which is located on tracks 0, 1) onto drive B. This may be accomplished by entering the following command at the keyboard:

A >**SYSGEN** <cr>

The SYSGEN command is used to read the operating system from a diskette and place it on the desired diskette. Once you have entered this command at the keyboard and typed **RETURN**, the disk operating system will ask you to select which drive you want to take the source from. The correct answer to this question is the letter "A". After entering **A** depress the **RETURN**.

The next question the program will ask is where do you want the source to be placed (the destination drive). The correct answer to this is the letter "B" indicating drive B. Once you have entered this, the operating system will be copied from drive A onto drive B.

After this process has been completed the operating system will ask whether you wish to make another copy or to reload the operating system. The correct response is to simply enter a **RETURN** which will reload the operating system.

Once the operating system has been reloaded, you may remove the master disk operating

system in drive A. Once this disk has been removed, store it in a safe place, as you may need it later to generate additional copies of the disk operating system and its programs.

At this point you should have removed the master disk from drive A. Now remove the copy from drive B and reinstall it in drive A and close the door on drive A. After you have completed this, depress the RED reset keys located on either side of the keyboard. This will reset the machine and reload the newly installed operating system from the new system diskette.

MODEL 20

For the CompuStar Model 20 VPU, the procedure differs since there is only drive A. The operating system diskette that contains the FORMAT program is first inserted in drive A. Next, the user enters the **FORMAT** command and depresses the **RETURN** key.

At this point, FORMAT requires the user to select which type of diskette is being formatted. The user should select **2**. Next, FORMAT will prompt the user to insert the blank diskette to be formatted into drive A. This accomplished by removing the Operating System Diskette from drive A and inserting the diskette to be formatted. Once this is done, the user should depress the **F** key to begin formatting.

The FORMAT program will display current track numbers from 0 through 69 for the CompuStar Model 20.

The user, upon completion of the formatting operation, should remove the formatted diskette from drive A and reinsert the operating system diskette into drive A.

Note: During the copy operation that follows, the operating system will ask the user to switch the diskettes in drive A as appropriate.

To copy the contents of the operating system diskette, enter:

A > **PIP B:=*.*[V]** <**cr**>

Notice that the B drive was specified. The operating system will then copy a portion of the operating system diskette into memory, ask the user to insert the "B" diskette into drive A, transfer the portion of the operating system diskette in main memory to the "B" diskette in drive A and then request the "A" diskette be reinserted into drive A.

The PIP utility then reads the next portion into memory, asks again that the "B" diskette be inserted in drive A, and proceeds with this process until the entirety of the "A" diskette has been copied to the "B" diskette (with the exception of tracks 0 and 1 that contain the actual operating system.)

Copying the contents of the operating system diskette on a Model 20 (single disk drive unit) is a lengthy process. Because the Model 30 and Model 20 diskette are identical in recording density and format, Model 20 diskettes may be copied (using PIP) on a Model 30. If a Model 30 is available, this can speed up the process considerably.

Next, the SYSGEN procedure is used to copy tracks 0 and 1 to the new operating system diskette that is being built. Enter the **SYSGEN** command and depress the **RETURN** key after replacing the original operating system diskette in drive A.

SYSGEN then asks which drive the operating system (source) is located on. Respond **A.** At this point, the diskette in A will be accessed and tracks 0 and 1 copied into memory. After the red "in-use" indicator on the disk drive goes out, remove the diskette from drive A and insert the diskette upon which the operating system is to be duplicated.

SYSGEN then asks the user to specify which drive the operating system is to be copied to. Here specify **A**, the drive into which the diskette being "built" was placed. The diskette in the A drive now contains a new copy of the operating system.

Depressing the red keys at this point reloads the operating system from the new operating system diskette. The original operating system diskette should be kept in a safe place in case it is needed again.

IMPORTANT: If random, garbled information is displayed on the screen at this time, this indicates that an error was made in the use of the SYSGEN program. If this is the case, remove the new diskette from drive A and reinstall the original master system diskette and repeat the previously outlined procedure for generating a new disk operating system. If you still encounter difficulties, please refer to the CP/M sections of this manual for more detailed information concerning the SYSGEN procedure.

Now that you have successfully completed the generation of a new system diskette, please refer to the CP/M sections of this manual for a complete description of all of the operating system utility programs (DDT.COM, PIP.COM, SUBMIT.COM, etc.).

OPTIONAL SOFTWARE

Microsoft Fortran 80 — comparable to Fortran compilers on large mainframes and minicomputers. All of ANSI standard Fortran X3.9-1966 is included except the COMPLEX datatype. Therefore, users may take advantage of the many application programs already written in Fortran. Fortran 80 is unique in that it provides a microprocessor Fortran and assembly language development package that generates relocatable object modules. This means that only the subroutines and system routines required to run Fortran 80 programs are loaded before execution. Subroutines can be placed in a system library so that users develop a common set of subroutines that are used in their programs. Also, if only one module of a program is changed, it is necessary to recompile **only** that module.

COMPUSTAR
STAND-ALONE
SOFTWARE SUMMARY

COMPUSTAR STAND-ALONE SOFTWARE SUMMARY

The software distributed with the CompuStar VPU is basically of two types. First, CP/M and miscellaneous software from Digital Research provide an operating system, and various utility programs. Second, there are utility programs prepared by Intertec for special features or functions of the CompuStar VPU and an interpreted BASIC from MicroSoft. A summary of both categories follows:

CP/M SUMMARY

PROGRAM NAME	FUNCTION	ENTRY EXAMPLE
PIP.COM	Copies files between devices, logical and physical.	**PIP B:=A:*.*⟨cr⟩** **PIP CON:=A:FILE.TYP⟨cr⟩**
SYSGEN.COM	Generates a new operating system on diskette.	**SYSGEN⟨cr⟩**
ED.COM	Text Editor, allows changes to text files.	**ED PROGRAM.ASM⟨cr⟩**
ASM.COM	Assembles an 8080-type assembly language that produces a source listing and a 'HEX' file.	**ASM PROG⟨cr⟩**
LOAD.COM	Creates a binary object file from a 'HEX' file that can be executed.	**LOAD PROG⟨cr⟩**
DDT.COM	Allows user to debug and step through a 'COM' or 'HEX' file's execution.	**DDT PROG.COM⟨cr⟩** **DDT PROG.HEX⟨cr⟩**
SUBMIT.COM	Performs successive execution of a list of 'COM' files.	**SUBMIT MORNING⟨cr⟩**
XSUB.COM	Forces data entry into a process under control of SUBMIT.	**XSUB⟨cr⟩**
DUMP.COM	Produces a hexadecimal listing of a disk file's contents.	**DUMP PROG.COM⟨cr⟩**
STAT.COM	Display file status, device status, or system characteristics.	**STAT B:*.*⟨cr⟩** **STAT B:DSK:⟨cr⟩**
DIR*	Displays a disk directory.	**DIR⟨cr⟩** **DIR B:⟨cr⟩**
ERA*	Erases a disk file.	**ERA B:PROG.BAK⟨cr⟩**
REN*	Renames a disk file.	**REN PROG.ASM=PROG⟨cr⟩**
SAVE*	Saves memory contents on the disk.	**SAVE 10 A.COM⟨cr⟩**
TYPE*	Displays an ASCII listing of a disk file's contents.	**TYPE PROG.PRN⟨cr⟩**

*These are CP/M command level functions

These programs or commands run under the CP/M 2.2 disk operating system (DOS). This DOS is customized for each CompuStar VPU computer model available, which results in having three operating systems applicable to the CompuStar VPU product line. These are:

* CS20CPM.COM — Model 20 VPU. The corresponding BIOS is CSE20BS.ASM.

* CS30CPM.COM — Model 30 VPU. The corresponding BIOS is CSE30BS.ASM.

* CS40CPM.COM — Model 40 VPU. The corresponding BIOS is CSE40BS.ASM.

The difference between these models of the CompuStar VPUs is the amount of on-board floppy disk storage each contains. The correct operating system is distributed with each VPU.

Refer to later sections of this manual for detailed documentation of CP/M usage and capabilities.

CompuStar Model 20 Stand-Alone Considerations

The CompuStar Model 20 VPU has only one floppy disk drive. In a stand-alone mode this presents a unique situation. To assist the file copy operation, a logical disk request system has been implemented.

This works as follows. When the system boots up, the user is on logical drive A. As long as all accesses are directed to logical drive A the system performs normally. If the user attempts to access drive B, the system stops and requests the logical disk B. The user then takes the current disk out of the drive and puts in his other diskette. The system notes which diskette was accessed last and as long as the accesses continue to occur on that disk, no prompts will occur. When the user or the user's program attempts to access the logical drive A again, another prompt will request that logical drive A be inserted.

INTERTEC UTILITY SUMMARY

Program Name	Function
CONFIGUR.COM	Establishes certain user selectable operating characteristics of the CompuStar VPU.
FORMAT.COM	Prepares previously unused diskettes for use in the CompuStar VPU disk drives by placing sector information on them.
HEXDUMP.COM	Generates an "Intel" hexadecimal format data stream from any binary object file in the CompuStar computer and outputs it to a port.
64KTEST.COM	Performs extensive memory testing for diagnostic purposes.
RX/TX.COM	A program pair that enables file transfers between two Intertec computers.
CSEDIT.COM	A program that allows the user to generate or modify an alternate character set.
CSDUMP.COM	A program that allows the user to generate printed output of the alternate character set built with CSEDIT for documentation purposes.
TIME.COM	A program that allows the time maintained by the real time clock to be set and/or displayed.

Program Name	Function
DATE.COM	A program that allows the date to be entered or displayed.
FCOPY.COM	A disk file copy utility for use with CompuStar Model 20 VPUs.
MBASIC.COM	An interpreted type BASIC.

Utility Descriptions

In general, the Intertec utility programs are self-documenting and designed for ease of use. To support this design further, documentation of these programs follows. The interpreted BASIC from MicroSoft is documented in a separate manual available from Intertec.

CONFIGUR.COM
This program enables the user to select various operating parameters for the CompuStar VPU. This feature allows flexibility in your computer's use. The parameters affect the MAIN and AUXILIARY ports, the AC line frequency, keypad assignments, audio and visual feedback, and disk verification. By allowing the user to change these parameters, a variety of peripheral devices can be used with your CompStar VPU.

The CONFIGUR program is menu-driven; the user selects the parameter to change, and then follows the instructions listed. To initiate the CONFIGUR command, type **CONFIGUR**⟨**cr**⟩ at the keyboard. CONFIGUR will then accept your commands for parameter changes. After you are finished, press the **RETURN** key (you may change several of the parameters if you wish); the screen will clear, and you will be instructed to press both RED keys on the keyboard. This action will force an operating system to reload containing your new parameters, and these parameters will be reloaded each time you reset the operating system.

Note: the CONFIGUR program is used in several different versions as different VPU applications allow different parameter sets to be configured. The CONFIGUR program will change the copy of the operating system located on the diskette in drive A. Even if your copy of CONFIGUR.COM is located on drive B, drive A will be affected. A summary of parameter selections is included for reference. This summary covers all options that can be configured, though some VPU applications do not allow certain options to be selected. Additionally, if the operating system diskette is write protected, the changes inputted with CONFIGUR will not be effected and the disk errors will occur.

Vertical Scan Frequency

The vertical scan frequency is selectable for 50 or 60 Hertz. This compensates for the local AC line frequency to prevent the display from flickering.

Disk Write Verification

You may select to have the Operating System perform disk read-back verification after each floppy disk write. This feature will 'double-check' the write operation.

Time Display Enable/Disable

If you wish for the time of day to be constantly displayed in the upper right corner of the screen upon power-up, you may select this feature here. Note that the time is always maintained internally, even if you choose not to display it. Also note that this setting is only for power-up, and you may select/deselect the time during operation by typing a Control-T (14H).

Key Click Enable/Disable
(Only operational in stand-alone mode)

You may choose to have the audible feedback feature enabled upon system power-up. Whenever the audible feedback is enabled, the computer will inform the operator with a slight 'click' at each key depression. Note that this setting is only for system power-up, and the key click feature can be changed during operation by typing a Control-B (02H).

Main and Aux Port Operation

Choosing these selections will permit you to change the operating parameters of the MAIN and AUX serial I/O ports located on the rear of your computer. The details of this selection are covered below including which ports are applicable for a given feature.

Operating Mode (MAIN Port Only)
(Only operational in stand-alone mode)

The MAIN port operating mode selections are synchronous and asynchronous. Be certain that the peripheral with which you are communicating is capable of operating in the same mode; they cannot be different. Note also that when changing to synchronous mode, you may need to change the number of SYNC Characters and the SYNC Character value. When changing to the asynchronous mode, you may need to change the number of stop bits. Using the synchronous mode requires different switch settings to be modified on the Keyboard/CPU module. Refer to the Synchronous Communication topic in this section for further information.

Baud Rate (MAIN and AUX Ports)

A wide range of baud rates can be selected for the port including rates from 9600 baud (approximately 960 characters/second) to 50 baud (5 characters/second). Select the baud rate needed to communicate with your peripheral.

Number of SYNC Characters (Main Port Only)

This selection will affect the number of SYNC Characters sent to the USART upon system power-up. Select either one or two.

Number of Stop Bits (MAIN and AUX Ports)

This selection will choose the number of stop bits sent after each character when the port is operating in asynchronous mode. Select either 1, 1.5, or 2 stop bits.

Character Length (MAIN and AUX Ports)

You may select the length of the character to be transmitted and received. Many selections are provided to insure compatibility with older TTY and Baudot machines. Usually, eight bits is the standard character length. You may, however, select 5, 6, 7, or 8 bit character lengths.

Parity (MAIN and AUX Ports)

You may choose to check parity with each transmission. This will provide a limited 'checksum' to help insure that proper transmission has occurred. However, if parity is enabled, the application program will have to test the USART status register for parity error. You may also select Even or Odd parity. If you choose to check parity, be certain that the device with which you are communicating matches your setting.

Handshaking (MAIN and AUX Ports)
(MAIN Port operational in stand-alone mode Only)

If you wish to check Data Set Ready prior to each character transmission, you should enable this function. This will permit a peripheral device to signal the computer whenever it cannot receive anymore characters.

SYNC Character Value (MAIN Port Only)
(Only operational in stand-alone mode)

The SYNC Character is the byte that is sent to the USART after it has been programmed for synchronous communication. Generally, the ASCII value of 13H (SYN) is used, but any binary value may be substituted. Make certain that the SYNC Character value matches that of the peripheral device with which you are communicating. Enter the hexadecimal number desired.

Keypad Reprogramming
(Only operational in stand-alone mode)

The 18 key numeric keypad on the right side of the keyboard can be reprogrammed to any input values desired. You may, for example, wish to invert the numeric keys on the pad. They will then correspond to 'telephone style' with 1-2-3 on the top row and 7-8-9 on the bottom. You may wish to replace the keys with control-codes which are accepted by a word processing or text editing program. The key cap values could then be changed to descriptive messages which are easier to learn and understand. Any value from OOH to FFH can now be assigned to the numeric keys with CONFIGUR. See the ASCII chart and control-code information in the Miscellaneous Operational Information section.

When this selection is entered, an image of the keypad appears on the screen. To change the value of any key, depress the **TAB** key until the cursor is over the key you wish to change. Then press the escape **ESC** key to indicate the change needed. The cursor will position itself on the last line, and a blinking asterisk will replace the cursor on the key being changed. Enter the new hexadecimal value for this key. Your input must be a valid hex number between O-F as invalid numbers will not be accepted. Press the **RETURN** key when you are finished.

To restore the keypad to its original values press the **R** key instead of the **ESC** or **TAB** keys. The screen will be updated instantly, and the cursor will be repositioned at the beginning of the display. When all changes have been entered, pressing the **RETURN** key (instead of the **ESC** or **TAB** keys) will return you to the main menu of selections.

FORMAT.COM

Before diskettes can be used by an Intertec computer, they must first be formatted. This process will erase the diskette of all data and write certain sector-header information on the diskette so that the operating system is able to properly locate data on the diskette. FORMAT.COM is a versatile program that will allow the user to format diskettes for the CompuStar VPUs.

For Model 30 and 40 VPUs

To load the format program from diskette, type **FORMAT** ⟨**cr**⟩ at the keyboard. After loading, you should select the type of diskette you wish to format. Once your selection has been entered, you will be asked to place an unformatted diskette into drive B and type the **F** key to begin formatting. When the formatting is completed, you may continue formatting by placing another diskette into drive B and pressing the **F** key. You may repeat this process until all of your diskettes have been formatted. Press the **RETURN** key to end the formatting session.

The diskette that you format does not have to be a blank diskette. You may format an old diskette if you wish, but you should remember that **FORMAT** will destroy all data on a diskette. However, if the data on a diskette becomes damaged (or if you suspect that the data is damaged), copy the diskette onto another diskette and reformat the original. This way, you save some (or all) of the original data and you don't lose any diskettes.

For Model 20 VPUs

For the CompuStar Model 20 VPU, the procedure differs since there is only drive A. The operating system diskette that contains the FORMAT program is first inserted in drive A. Next, the user enters the **FORMAT** command and depresses the **RETURN** key.

At this point, FORMAT requires the user to select which type of diskette is being formatted. The user should select **2.** Next, FORMAT will prompt the user to insert the blank diskette to be formatted into drive A. This is accomplished by removing the Operating System Diskette from drive A and inserting the diskette to be formatted. Once this is done, the user should depress the **F** key to begin formatting.

The FORMAT program will display current track numbers from 0 through 69 for the CompuStar Model 20.

The user, upon completion of the formatting operation, should remove the formatted diskette from drive A and reinsert the operating system diskette into drive A.

HEXDUMP.COM

This is a utility designed to convert a **COM** file to the Intel Hex format and transmit it from the Aux or MAIN port to a desired port. Since the PIP program cannot transfer **COM** files, this utility is useful in effecting file transfers without the PIP program. To initiate the HEXDUMP facility, type the following at the keyboard: **HEXDUMP**⟨**cr**⟩. The program will be loaded and then await your instructions.

The first thing that the HEXDUMP procedure requests is the port to which you wish to dump the file. Here enter **1** for the MAIN port (corresponding to CP/M's PUN: and RDR: device), or **2** for the AUXILIARY port (corresponding to CP/M's LST: device). You must enter either a **1** or **2**; invalid entries will be ignored. Next you may choose whether or not you wish to have the **HEX** file echoed to the console (this will display the file as transmitted). Enter **1** if you do not wish to have the file echoed on the screen, or **2** if you wish to have the contents echoed. Again, invalid entries will be ignored.

Now you are ready to enter the file name. You must enter the drive designator, the file name and the file type. Separate the drive indicator from the file name with a colon (**:**), and separate the type from the name with a period (**.**). Press the **RETURN** key after entering the name.

Example:

A⟩ **HEXDUMP**⟨**cr**⟩

HEXDUMP FILE UTILITY VER. 3.1

SELECT ONE OF THE FOLLOWING: (TYPE THE NUMBER)
1 — THE MAIN PORT (PUN:)
2 — THE AUX PORT (LST:)

2

SELECT ECHO ON THE CONSOLE:
1 — DO NOT ECHO ON THE CONSOLE
2 — ECHO TO THE CONSOLE

1

ENTER DISC, FILE-NAME, AND FILE-TYPE TO BE TRANSFERRED.

A:STAT.COM⟨**cr**⟩

FILE TRANSFER COMPLETED.

In the example above, the file STAT.COM was transferred from disk A through the auxiliary port.

HEXDUMP.COM will only transfer files which exist on drives A and B. If you enter an erroneous file-name or disk drive, the program will display an error message. If HEXDUMP.COM is unable to locate the given file, another error message will be given. When the transmission has completed, the screen will indicate this and return to the operating system.

64KTEST.COM

This program performs an extensive test on main memory by writing and reading all possible binary patterns to all locations in the random access memory (RAM). The process takes between eight and ten minutes to complete.

The test procedure begins by typing **64KTEST**⟨**cr**⟩ at the keyboard. After the program is loaded into memory, you will be asked to remove all diskettes from their drives. If you have a Hard Disk Storage System connected to the terminal to be tested, either power down the hard disk or disconnect it from the terminal by removing the interconnecting cable. Be sure the Key Click feature is turned off before running the 64KTEST program. Otherwise, errors will be indicated that do not exist.

Once you have pressed the **G** key to start the test, the screen should fill with random text. The patterns on the screen should move around. This is because the memory for the screen is also undergoing the test. After the test is completed, the screen will display RAM OK, indicating that the test was successful. The test is an endless loop, and will repeat until the RED keys are depressed simultaneously. Therefore, you can test the RAM as long as you desire.

If an error is detected by the test, the test will stop and the audible tone will sound continuously. Should this occur, re-try the test. If the error occurs frequently, please contact Intertec Customer Services Department.

TX.COM

The TX utility is written in standard CP/M assembly language. TX is designed to communicate via the computer's Main Port with the program RX running in the destination machine. Therefore, TX is considered the "Master" program, and RX is the "Slave" program. RX receives commands from TX such as "Open file", "Read incoming data block", "Write block to file", and so on. For this reason, the user should only be concerned with console operations for the machine in which TX is running. RX receives all directions from the communications link.

Unlike data transfer operations initiated with PIP, the TX/RX pair perform block verification and re-transmission in the event of error. TX/RX may be used to send any type of CP/M file without modification including .COM files.

TX is initiated by typing the command/ **TX**⟨**cr**⟩. The TX program will then "sign-on" with an identifying message and version number and then give the user an option to proceed or abort. The actual console dialogue appears as:

A ⟩ **TX**⟨**cr**⟩

INTERTEC File Transfer Utility Vers 1.X
HIT CR WHEN RECEIVE MACHINE READY OR Q TO ABORT

At this point, start up RX in the destination machine (See the RX.COM description that follows this TX description).

When a carriage return is entered to TX, it will attempt to establish a linkage to the destination RX machine over the computer's Main Port. Given that a link can be established, TX will display the message:

LINK TO SLAVE MACHINE ESTABLISHED

or, if many attempts to link fail:

UNABLE TO ESTABLISH/MAINTAIN DATA LINK

(This probably indicates that some aspect of the connection with the destination machine is not correct, i.e. inconsistent baud rates, improper cabling, or excessive line noise.)

The TX program then prompts the user to enter both the source file name and the destination file name. These names must be fully qualified, non-ambiguous file references. This includes disk specifiers.

If the specified file already exists on the receiving machine, TX will display:

FILE ALREADY EXISTS ON RECEIVING MACHINE

and the link is terminated.

As an expediency, send the file again, but with a temporary destination file name.

As a file is being transmitted under TX/RX, both TX and RX will display a record count. This serves to indicate that the data is being transferred correctly. It is normal to see a difference of one record between the two counts upon completion of a file transfer.

If TX detects a failure in the data link, it will output the message:

UNABLE TO ESTABLISH/MAINTAIN DATA LINK

When a file has been transmitted, TX displays the message:

FUNCTION COMPLETE
TYPE R TO REPEAT, CR TO EXIT

If another file is to be transferred, enter the letter **R** and TX will request another pair of file names. Entry of a carriage return will cause TX to command RX to shutdown and both will terminate.

There are two other messages that could be output by TX.

As each data block is sent, a checksum is calculated and transmitted. If RX detects a discrepancy between the received checksum and that which has been calculated for the received data, it will request that TX re-send the block in question. If the block cannot be received correctly after several re-transmissions, the message:

HARD DATA TRANSMISSION ERROR

will be rendered. The most likely cause of this failure is hardware error.

If the diskette on which RX attempts to place the incoming data file is write protected, or if there is not enough space to contain the incoming file, TX will display:

RECEIVE CANNOT CLOSE FILE

RX.COM

RX is an assembly language program designed to receive data files transmitted by TX from the computer's Main Port. It operates as a slave to the TX program, receiving commands from TX to perform operations on the destination machine.

RX is initiated by entering the command

A>**RX**<**cr**>

upon initiation, RX displays a "sign-on" message of the form:

INTERTEC File Transfer Utility Vers 1.X

From this point on, unless an error condition occurs, no further operator action is required.

As each data block is received, RX outputs a running count of the data blocks received. At the end of each received file, RX displays the message:

END-OF-FILE RECEIVED

When all files have been received, TX will command RX to terminate and RX will display:

LINK TERMINATED

If the data link cannot be established or maintained (indicated by a message on the TX system), it will be necessary to reset the destination system. This is accomplished on the destination computer by depressing both RED keys simultaneously.

TIME

The TIME program is used to set or display the time data kept by the real time clock. To set the time, enter:

A > **TIME hh:mm (AM)(PM)**<**cr**>

To enter "military" time (0000 thru 2400), it is not necessary to enter AM or PM. Once the entry is made, the TIME program will request that any key be depressed to set the time. This allows the user an opportunity to synchronize the time with another timepiece. To display the time, enter:

A > **TIME**<**cr**>

DATE

The DATE program is used to set or display the date maintained by the TIME/DATE clock. To set the date, follow the sample entries below:

A >**DATE 04/07/82 WED**<cr>

or

A > **DATE 04/07/82 WEDNESDAY**<cr>

To display the date, enter:

A > **DATE**<cr>

FCOPY

This utility is designed to simplify the file copy process necessary in single floppy disk drive VPU operations. To use FCOPY, enter:

A > **FCOPY [d:] ufn-destination [d:] ufn-source** <cr>

where d: is the appropriate destination or source drive designation. UFN is an unambiguous file name reference for the destination files name or the source files name as appropriate.

FCOPY prompts the user to switch diskettes in drive A as appropriate.

SECONDARY CHARACTER SET OPTION

As was stated in the theory of operation section, the CompuStar VPU provides a means by which a secondary character set option may be added. This gives the user the ability, via the software, to select either set. Intertec will provide a limited number of these alternate character sets, or if required, the customer may create a character set using software that is supplied by Intertec. In the following sections, both of these methods will be explained.

Intertec Furnished Secondary Character Sets

The easiest and quickest way to have access to a secondary character set would be to purchase one of the sets available from Intertec. This character set would be contained on an EPROM that would be inserted into a vacant IC socket on the processor board. After the EPROM has been inserted into its socket, it can be initialized via the escape sequence given in the attribute program section. Secondary character set installation procedures will be provided with each set purchased from Intertec.

Customer Created Secondary Character Sets

For those requirements where Intertec does not offer a suitable secondary character set, one can be created by the user. The CP/M disk provided with the CompuStar contains two utility programs that provide the means for creating and verifying secondary character sets. These two programs are CSEDIT.COM and CSDUMP.COM.

CSEDIT.COM

The CSEDIT utility provides the means for creating a secondary character set. The program is loaded from the disk by entering:

A>**CSEDIT** <cr>

The initial screen message will read:

CompuStar Character Set Editor — Ver 1.X

Enter the character set file name:

The new character set file name should then be entered in the normal format of **filename.typ** and then pressing **RETURN**. The next screen message will read:

Enter hex value of character to edit (0-7F, eXit, Quit, or ?)

As indicated by the parenthesis, there are four options (0-7F, eXit, Quit, or ?) available at this point. Since the "?" is the help page and will explain the other 3 entries, type **?** and press **RETURN**. The following page will appear on the screen:

The input required at this point is the hex value of the ASCII character that you wish to edit. This value must be in the range of 00 to 7F hex. You may also enter a **X** to exit the program and update the character set file, or a **Q** to abort the program and not update the character set file.

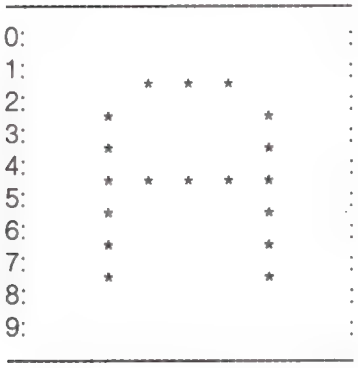
"0" — Clear dot at current position
" " — Put dot at current position
ENT — Go to start of next line
"-" — Clear current line
"1" — Invert pattern dots
"2" — Save pattern in temp. buffer
"3" — Recall previously saved pattern
"7" — Clear character cell
ESC — End editing of character
BRK — Abort with no change to character

All cursor keys on the keypad work as would be expected.

Hit RETURN to continue:

After reading the help page, pressing **RETURN** will cause the initial screen message of the program to reappear. At this time the user should be ready to start the process to create his alternate character set. The following examples are from the standard character set provided with the CompuStar VPU.

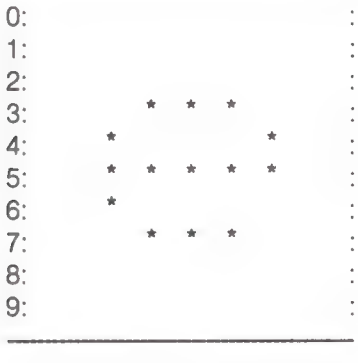
Editing Number - 41H



"." — Put dot at current position
"_" — Clear current line
"1" — Invert pattern dots
"3" — Recall previously saved pattern
ESC — End editing of character

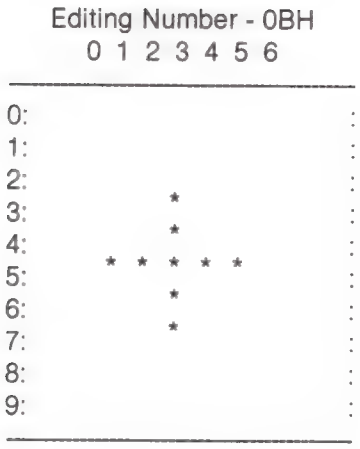
"0" — Clear dot at current position
ENT — Go to start of next line
"2" — Save pattern in temp. buffer
"7" — Clear character cell
BRK — Abort with no change to pattern

Editing Number - 65H



"." — Put dot at current position
"_" — Clear current line
"1" — Invert pattern dots
"3" — Recall previously saved pattern
ESC — End editing of character

"0" — Clear dot at current position
ENT — Go to start of next line
"2" — Save pattern in temp. buffer
"7" — Clear character cell
BRK — Abort with no change to pattern



"." — Put dot at current position
"_" — Clear current line
"1" — Invert pattern dots
"3" — Recall previously saved pattern
ESC — End editing of character
"0" — Clear dot at current position
ENT. — Go to start of next line
"2" — Save pattern in temp. buffer
"7" — Clear character cell
BRK — Abort with no change to pattern

After all the secondary characters have been created, by typing **X** and pressing **RETURN**, the new character set will be written on the disk as a binary file and the verification process can begin.

CSDUMP.COM

The CSDUMP utility will be used to verify that the character set that was just created is what is needed. To run the CSDUMP program, insure the CompuStar is connected to a printer via the Auxiliary port. The printer is the only output device that will display the dump. Once this is accomplished, enter

A>**CSDUMP** <cr>

The following message will appear on the screen.

CompuStar Character Set Dump — Ver. 1.X

Enter character set file name:

Enter the filename and press the **RETURN** key. The character set will be dumped out to the printer and the resulting page set should look similar to the one shown on the Sample Page — Character Set Dump exhibit.

The character file generated by this procedure can then be transferred to an EPROM programming machine using the HEXDUMP.COM utility. Once the EPROM has been created, it should then be inserted into the empty IC socket Z75, as indicated in the Socket Z75 figure. The initialization of the new character set is contained in the Escape Sequence covered in the Attribute section of this manual.

The part numbers for the blank EPROM are, Intertec part number 30122516 or Texas Instruments part number TMS-2516JL-35 or equivalent.

Any questions concerning Intertec created secondary character sets or the procedures or materials necessary to create secondary character sets should be referred to the Customer Services Department at Intertec Data Systems Corporate Headquarters.

SAMPLE PAGE — CHARACTER SET DUMP EXHIBIT

CompuStar Character Set Dump
File Name : A:STDSET

```
0:
1:     * * *
2:   *       *
3:   *   * * *
4:   *   *   *
5:   *   * * *
6:   *
7:     * * *
8:
9:
   0 1 2 3 4 5 6
 ASCII value - 40h
```

```
0:
1:     * * *
2:   *       *
3:   *       *
4:   * * * * *
5:   *       *
6:   *       *
7:   *       *
8:
9:
   0 1 2 3 4 5 6
 ASCII value - 41h
```

```
0:
1:   * * * *
2:     *     *
3:     *     *
4:     * * *
5:     *     *
6:     *     *
7:   * * * *
8:
9:
   0 1 2 3 4 5 6
 ASCII value - 42h
```

```
0:
1:     * * *
2:   *       *
3:   *
4:   *
5:   *
6:   *       *
7:     * * *
8:
9:
   0 1 2 3 4 5 6
 ASCII value - 43h
```

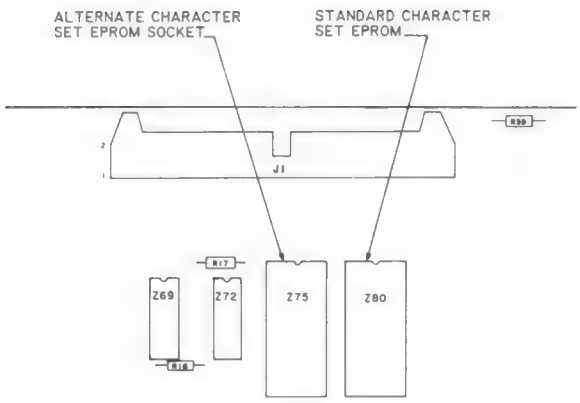
```
0:
1:   * * * *
2:     *     *
3:     *     *
4:     *     *
5:     *     *
6:     *     *
7:   * * * *
8:
9:
   0 1 2 3 4 5 6
 ASCII value - 44h
```

```
0:
1:   * * * * *
2:   *
3:   *
4:   * * * *
5:   *
6:   *
7:   * * * * *
8:
9:
   0 1 2 3 4 5 6
 ASCII value - 45h
```

```
0:
1:   * * * * *
2:   *
3:   *
4:   * * * *
5:   *
6:   *
7:   *
8:
9:
   0 1 2 3 4 5 6
 ASCII value - 46h
```

```
0:
1:     * * * *
2:   *
3:   *
4:   *     * * *
5:   *         *
6:   *         *
7:     * * * *
8:
9:
   0 1 2 3 4 5 6
 ASCII value - 47h
```

SOCKET Z75


ALTERNATE CHARACTER
SET EPROM SOCKET
STANDARD CHARACTER
SET EPROM
J1
Z69
Z72
Z75
Z80

MISCELLANEOUS OPERATIONAL INFORMATION

MISCELLANEOUS OPERATIONAL INFORMATION

USING THE "INP:" AND "OUT:" FEATURES OF PIP

Files can be transferred using the PIP program as described in the CompuStar manual section entitled 'An Introduction to CP/M Features and Facilities.' The CompuStar VPU is equipped with two RS232C Serial interface ports (labeled 'MAIN' and 'AUX' on the rear panel). Whenever the CompuStar transmits serial data via the 'MAIN' port, the destination is designated as a punch (PUN:); when receiving, the data source device is considered a reader (RDR:). When transmitting data to the 'AUX' port, the destination device is considered a list (LST:).

The 'MAIN' serial port may also be considered as an input (INP:) or output (OUT:) device. When used in this mode, the operator has the option of communicating with the sending or receiving device prior to file transfer by means of the CompuStar VPU console. This interface is factory programmed for the following operational mode:

Asynchronous Communication
1200 Baud Rate
8 Bit Character Length
1 Stop Bit
No Parity
DSR Disabled

Files transferred via the 'MAIN' port must be in Intel 'HEX' or ASCII format. BASIC source programs must be saved in ASCII format before they can be transferred. Binary files (i.e., programs) must be transferred as HEX files, using the program HEXDUMP.COM.

PLEASE NOTE THE FOLLOWING:

1) Connect the CompuStar VPU 'MAIN' port to the console input of the host computer. Make certain that the host computer and the CompuStar are sending and receiving data in a compatible fashion (i.e., baud rate, character length, et.al.).

2) The largest file that can be transferred by PIP is 25K. If files are larger than 25K, they must be broken down into smaller segments of 25K or less.

3) Binary files (or .COM files) cannot be transferred via the serial ports using PIP. The DOS Diskette supplied with your CompuStar VPU includes two facilities for binary file transfer. See TX/RX and HEXDUMP for more information.

4) The Clear-to-Send (CTS - Pin 5) line on the 'MAIN' port must be high (logical '1') before the CompuStar VPU will send data through this port. Insure that these signals are properly connected between CompuStar and the host computer.

5) The 'MAIN' port is arranged so that the CompuStar VPU appears as a processor rather than a terminal. If it is to be used as a terminal, pins 2 and 3 in the RS-232-C cable must be interchanged.

The following represents a sample file transfer session. Please note that bold characters are those typed by the operator, and the symbol **⟨cr⟩** means the **RETURN** key.

A. Transfer an ASCII file from CompuStar VPU to host computer:

(File name is ABC.FIL)

A⟩**PIP OUT: = ABC.FIL⟨cr⟩**
ECHO (Y/N) **Y**

NOTE — The CompuStar VPU will now perform as a terminal for the host computer. If you wish, you may transmit a line of text to the host computer before the file ABC.FIL is actually transferred. Anything typed at the console will be sent to the host computer. To initiate the file transfer, type Control-B.

CTRL-B (Hold down the CTRL Key, then 'B')

The file will be transferred, and should be displayed on the screen. Upon completion, PIP will exit and return to the operating system. When finished, it is necessary to signal end-of-file for the host computer. This is best done by using the EOF: facility of PIP:

A>**PIP OUT:=EOF:**<cr>
ECHO (Y/N) **Y**
+
CTRL B (Hold down the CTRL key, then 'B')

NOTE — The EOF presumes that the target machine uses a hex 1A (CTRL-Z) to indicate end of file.

The file transfer is now complete.

B. Transfer an ASCII file to the CompuStar VPU from the host computer:

(File name is ABC.FIL)

A>**PIP ABC.PRN = INP:**<cr>
ECHO (Y/N) **Y**

The CompuStar VPU is now ready to receive input from the host computer. Any further console entry at the CompuStar VPU will be sent to the host computer. If the host computer does not send an end-of-file character, it will be necessary for you to place one into the file. This is done with the following command:

CTRL-Z (Hold down the CTRL key, then 'Z')
End of File, Control Z? (The computer asks for confirmation)
CTRL-Z (Hold down the CTRL key, then 'Z')

C. Transfer a Binary (or COM) file.

PIP does not permit binary files to be transferred via the serial port. Two system utilities, HEXDUMP and TX/RX, are provided to facilitate this. HEXDUMP will convert a binary file into a HEX format, and transmit out the 'MAIN' port. If HEXDUMP is used, the receiving unit must use PIP to accept the input from the sending unit. After the file transfer, the file can be converted back into a binary file using the DDT system program or the LOAD system program.

SYNCHRONOUS COMMUNICATION

Your computer system is factory configured to program the Universal Synchronous/Asynchronous Receiver/Transmitter (USART) to operate in the asynchronous mode. It is possible, however, to change this and permit the Bisync, or byte synchronous, communication mode. You will be responsible for writing the software drivers that send and receive synchronous data through to the MAIN port at the rear of your terminal. This section will instruct you to properly program the USART which is the interface between the CPU and the main port of your computer.

Before proceeding, it would be helpful to read the specifications sheets for the 8251-type USART. On these sheets you are given the control words to reprogram the USART.to enable synchronous communication. It is important that the timing dipswitch, located on the processor board, be properly set. This is necessary to coordinate the clock pulses between the two terminals communicating in the synchronous mode.

The CompuStar VPU stores the command byte for the 8251 USART in memory. To use a different type of communication, several steps are necessary. The USART command word must be changed in order to change the USART's operating mode. The operating system must also be prevented from resetting the USART during an interrupt cycle.

CompuStar VPU Serial Communications DIP Switch

The serial communication DIP switch is located on the Keyboard/CPU printed circuit board inside the cabinet. It is accessed by removing the four screws from the bottom of the base that holds the cover in place. Next, make sure that the disk drive doors are closed, then lift off the cover. This will expose the Keyboard/CPU Module. The DIP switch is a five position switch on the top edge of the Keyboard/CPU Module. It is the only user settable switch on this module.

NOTE: When completing the procedures above, you may encounter a warranty certification seal. The seal will be positioned over one of the four bottom cover screws and clearly displays the warning, WARRANTY IS VOID IF LABEL REMOVED. This seal should not be removed if you intend to participate in any of Intertec's Satisfaction Assurance programs. Once this seal has been removed, the unit no longer qualifies for participation within these programs. For additional information concerning Intertec's Satisfaction Assurance programs, contact Intertec's Customer Services Department.

For the normal mode (*asynchronous communication mode), these switches should be set as follows:

1 — OFF, 2 — OFF, 3 — ON, 4 — ON, 5 — OFF

For the synchronous communication mode with another unit providing the transmitter and receiver clock, the switches should be set as follows:

1 — ON, 2 — ON, 3 — OFF, 4 — OFF, 5 — OFF

Listed below is a brief description of the function of each of these switches:

1— External Clock to transmitter section of MAIN USART — originates from PIN #15 on MAIN RS232 connector at rear of terminal.

2— External Clock to receiver section of MAIN USART — originates from PIN #17 on MAIN RS232 connector at rear of terminal.

3— Internal TX Clock to MAIN USART — When on, this switch enables the built-in baud rate generator (Western Digital BR-1941).

NOTE: When this switch is in the 'ON' position, switch 1 **MUST** be in the 'OFF' position.

*THE SWITCHES WERE SET FOR THE ASYNCHRONOUS COMMUNICATION MODE BEFORE SHIPPING FROM THE FACTORY

8251A/S2657
PROGRAMMABLE COMMUNICATION INTERFACE

- **Synchronous and Asynchronous Operation**
- **Synchronous 5-8 Bit Characters; Internal or External Character Synchronization; Automatic Sync Insertion**
- **Asynchronous 5-8 Bit Characters; Clock Rate—1, 16 or 64 Times Baud Rate; Break Character Generation; 1, 1½, or 2 Stop Bits; False Start Bit Detection; Automatic Break Detect and Handling**
- **Synchronous Baud Rate — DC to 64K Baud**
- **Asynchronous Baud Rate — DC to 19.2K Baud**
- **Full Duplex, Double Buffered, Transmitter and Receiver**
- **Error Detection — Parity, Overrun and Framing**
- **Fully Compatible with 8080/8085 CPU**
- **28-Pin DIP Package**
- **All Inputs and Outputs are TTL Compatible**
- **Single +5V Supply**
- **Single TTL Clock**

The Intel® 8251A is the enhanced version of the industry standard, Intel® 8251 Universal Synchronous/Asynchronous Receiver/Transmitter (USART), designed for data communications with Intel's new high performance family of microprocessors such as the 8085. The 8251A is used as a peripheral device and is programmed by the CPU to operate using virtually any serial data transmission technique presently in use (including IBM "bi-sync"). The USART accepts data characters from the CPU in parallel format and then converts them into a continuous serial data stream for transmission. Simultaneously, it can receive serial data streams and convert them into parallel data characters for the CPU. The USART will signal the CPU whenever it can accept a new character for transmission or whenever it has received a character for the CPU. The CPU can read the complete status of the USART at any time. These include data transmission errors and control signals such as SYNDET, TxEMPTY. The chip is constructed using N-channel silicon gate technology.

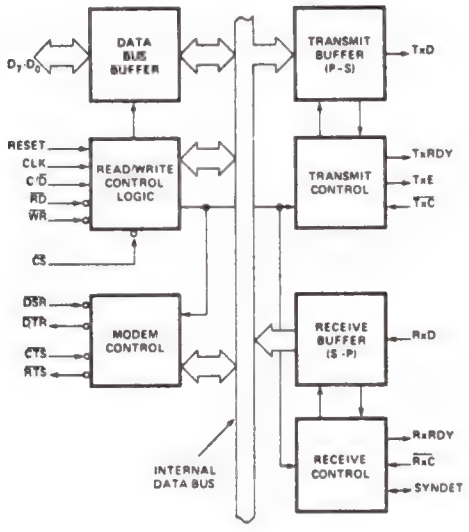


Figure 1. Block Diagram

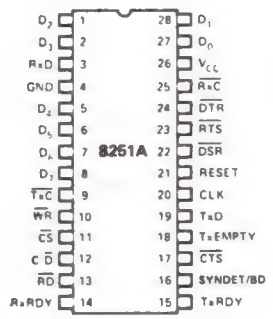


Figure 2. Pin Configuration

FEATURES AND ENHANCEMENTS

8251A is an advanced design of the industry standard USART, the Intel® 8251. The 8251A operates with an extended range of Intel microprocessors that includes the new 8085 CPU and maintains compatibility with the 8251. Familiarization time is minimal because of compatibility and involves only knowing the additional features and enhancements, and reviewing the AC and DC specifications of the 8251A.

The 8251A incorporates all the key features of the 8251 and has the following additional features and enhancements:

- 8251A has double-buffered data paths with separate I/O registers for control, status, Data In, and Data Out, which considerably simplifies control programming and minimizes CPU overhead.
- In asynchronous operations, the Receiver detects and handles "break" automatically, relieving the CPU of this task.
- A refined Rx initialization prevents the Receiver from starting when in "break" state, preventing unwanted interrupts from a disconnected USART.
- At the conclusion of a transmission, TxD line will always return to the marking state unless SBRK is programmed.
- Tx Enable logic enhancement prevents a Tx Disable command from halting transmission until all data previously written has been transmitted. The logic also prevents the transmitter from turning off in the middle of a word.
- When External Sync Detect is programmed, Internal Sync Detect is disabled, and an External Sync Detect status is provided via a flip-flop which clears itself upon a status read.
- Possibility of false sync detect is minimized by ensuring that if double character sync is programmed, the characters be contiguously detected and also by clearing the Rx register to all ones whenever Enter Hunt command is issued in Sync mode.
- As long as the 8251A is not selected, the $\overline{RD}$ and $\overline{WR}$ do not affect the internal operation of the device.
- The 8251A Status can be read at any time but the status update will be inhibited during status read.
- The 8251A is free from extraneous glitches and has enhanced AC and DC characteristics, providing higher speed and better operating margins.
- Synchronous Baud rate from DC to 64K.
- Fully compatible with Intel's new industry standard, the MCS-85.

FUNCTIONAL DESCRIPTION

General

The 8251A is a Universal Synchronous/Asynchronous Receiver/Transmitter designed specifically for the 80/85 Microcomputer Systems. Like other I/O devices in a Microcomputer System, its functional configuration is programmed by the system's software for maximum flexibility. The 8251A can support virtually any serial data technique currently in use (including IBM "bi-sync").

In a communication environment an interface device must convert parallel format system data into serial format for transmission and convert incoming serial format data into parallel system data for reception. The interface device must also delete or insert bits or characters that are functionally unique to the communication technique. In essence, the interface should appear "transparent" to the CPU, a simple input or output of byte-oriented system data.

Data Bus Buffer

This 3-state, bidirectional, 8-bit buffer is used to interface the 8251A to the system Data Bus. Data is transmitted or received by the buffer upon execution of INput or OUTput instructions of the CPU. Control words, Command words and Status information are also transferred through the Data Bus Buffer. The command status and data in, and data out are separate 8-bit registers to provide double buffering.

This functional block accepts inputs from the system Control bus and generates control signals for overall device operation. It contains the Control Word Register and Command Word Register that store the various control formats for the device functional definition.

RESET (Reset)

A "high" on this input forces the 8251A into an "Idle" mode. The device will remain at "Idle" until a new set of control words is written into the 8251A to program its functional definition. Minimum RESET pulse width is 6 t_{CY} (clock must be running).

CLK (Clock)

The CLK input is used to generate internal device timing and is normally connected to the Phase 2 (TTL) output of the 8224 Clock Generator. No external inputs or outputs are referenced to CLK but the frequency of CLK must be greater than 30 times the Receiver or Transmitter data bit rates.

$\overline{WR}$ (Write)

A "low" on this input informs the 8251A that the CPU is writing data or control words to the 8251A.

$\overline{RD}$ (Read)

A "low" on this input informs the 8251A that the CPU is reading data or status information from the 8251A.

C/$\overline{D}$ (Control/Data)

This input, in conjunction with the $\overline{WR}$ and $\overline{RD}$ inputs, informs the 8251A that the word on the Data Bus is either a data character, control word or status information.
1 = CONTROL/STATUS 0 = DATA

$\overline{CS}$ (Chip Select)

A "low" on this input selects the 8251A. No reading or writing will occur unless the device is selected. When $\overline{CS}$ is high, the Data Bus in the float state and $\overline{RD}$ and $\overline{WR}$ will have no effect on the chip.

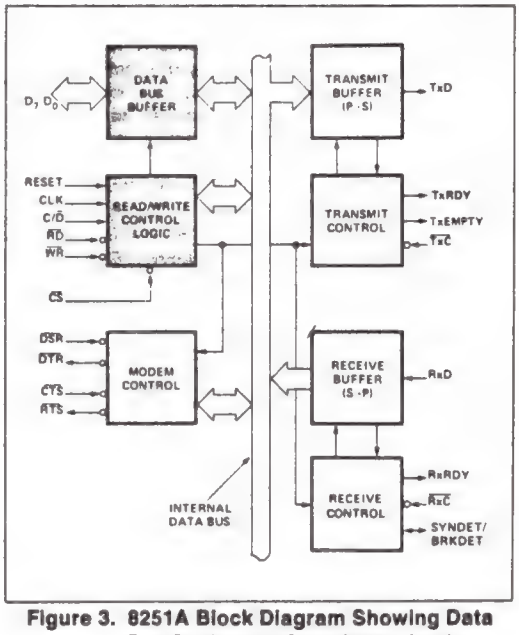


Figure 3. 8251A Block Diagram Showing Data Bus Buffer and Read/Write Logic Functions

C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	
0	0	1	0	8251A DATA → DATA BUS
0	1	0	0	DATA BUS → 8251A DATA
1	0	1	0	STATUS → DATA BUS
1	1	0	0	DATA BUS → CONTROL
X	1	1	0	DATA BUS → 3-STATE
X	X	X	1	DATA BUS → 3-STATE

Modem Control

The 8251A has a set of control inputs and outputs that can be used to simplify the interface to almost any Modem. The Modem control signals are general purpose in nature and can be used for functions other than Modem control, if necessary.

$\overline{DSR}$ (Data Set Ready)

The $\overline{DSR}$ input signal is a general purpose, 1-bit inverting input port. Its condition can be tested by the CPU using a Status Read operation. The $\overline{DSR}$ input is normally used to test Modem conditions such as Data Set Ready.

$\overline{DTR}$ (Data Terminal Ready)

The $\overline{DTR}$ output signal is a general purpose, 1-bit inverting output port. It can be set "low" by programming the appropriate bit in the Command Instruction word. The $\overline{DTR}$ output signal is normally used for Modem control such as Data Terminal Ready or Rate Select.

$\overline{RTS}$ (Request to Send)

The $\overline{RTS}$ output signal is a general purpose, 1-bit inverting output port. It can be set "low" by programming the appropriate bit in the Command Instruction word. The $\overline{RTS}$ output signal is normally used for Modem control such as Request to Send.

$\overline{CTS}$ (Clear to Send)

A "low" on this input enables the 8251A to transmit serial data if the Tx Enable bit in the Command byte is set to a "one." If either a Tx Enable off or CTS off condition occurs while the Tx is in operation, the Tx will transmit all the data in the USART, written prior to Tx Disable command before shutting down. On the 8251A/S2657 if CTS off or Tx Enable off condition occurs before the last character written appears in the serial bit stream, that character will be transmitted again upon CTS on or Tx Enable on condition.

Transmitter Buffer

The Transmitter Buffer accepts parallel data from the Data Bus Buffer, converts it to a serial bit stream, inserts the appropriate characters or bits (based on the communication technique) and outputs a composite serial stream of data on the TxD output pin on the falling edge of $\overline{TxC}$. The transmitter will begin transmission upon being enabled if $\overline{CTS}$ = 0. The TxD line will be held in the marking state immediately upon a master Reset or when Tx Enable/$\overline{CTS}$ off or TxEMPTY.

Transmitter Control

The transmitter Control manages all activities associated with the transmission of serial data. It accepts and issues signals both externally and internally to accomplish this function.

TxRDY (Transmitter Ready)

This output signals the CPU that the transmitter is ready to accept a data character. The TxRDY output pin can be used as an interrupt to the system, since it is masked by Tx Disabled, or, for Polled operation, the CPU can check TxRDY using a Status Read operation. TxRDY is automatically reset by the leading edge of $\overline{WR}$ when a data character is loaded from the CPU.

Note that when using the Polled operation, the TxRDY status bit is *not* masked by Tx Enabled, but will only indicate the Empty/Full Status of the Tx Data Input Register.

TxE (Transmitter Empty)

When the 8251A has no characters to transmit, the TxEMPTY output will go "high". It resets automatically upon receiving a character from the CPU if the transmitter is enabled. TxEMPTY can be used to indicate the end of a transmission mode, so that the CPU "knows" when to "turn the line around" in the half-duplexed operational mode.

In SYNChronous mode, a "high" on this output indicates that a character has not been loaded and the SYNC character or characters are about to be or are being transmitted automatically as "fillers". TxEMPTY does not go low when the SYNC characters are being shifted out.

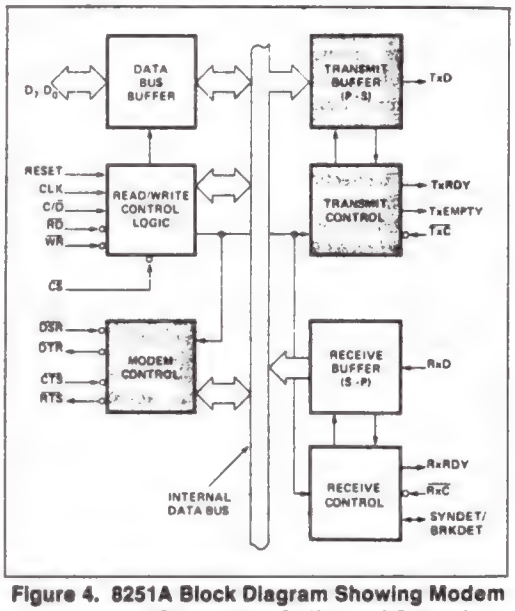


Figure 4. 8251A Block Diagram Showing Modem and Transmitter Buffer and Control Functions

$\overline{TxC}$ (Transmitter Clock)

The Transmitter Clock controls the rate at which the character is to be transmitted. In the Synchronous transmission mode, the Baud Rate (1x) is equal to the $\overline{TxC}$ frequency. In Asynchronous transmission mode the baud rate is a fraction of the actual $\overline{TxC}$ frequency. A portion of the mode instruction selects this factor; it can be 1, 1/16 or 1/64 the $\overline{TxC}$.

For Example:

If Baud Rate equals 110 Baud,
$\overline{TxC}$ equals 110 Hz (1x)
$\overline{TxC}$ equals 1.76 kHz (16x)
$\overline{TxC}$ equals 7.04 kHz (64x).

The falling edge of $\overline{TxC}$ shifts the serial data out of the 8251A.

AFN-01573B

Receiver Buffer

The Receiver accepts serial data, converts this serial input to parallel format, checks for bits or characters that are unique to the communication technique and sends an "assembled" character to the CPU. Serial data is input to RxD pin, and is clocked in on the rising edge of $\overline{RxC}$.

Receiver Control

This functional block manages all receiver-related activities which consist of the following features:

The RxD initialization circuit prevents the 8251A from mistaking an unused input line for an active low data line in the "break condition". Before starting to receive serial characters on the RxD line, a valid "1" must first be detected after a chip master Reset. Once this has been determined, a search for a valid low (Start bit) is enabled. This feature is only active in the asynchronous mode, and is only done once for each master Reset.

The False Start bit detection circuit prevents false starts due to a transient noise spike by first detecting the falling edge and then strobing the nominal center of the Start bit (RxD = low).

The Parity Toggle F/F and Parity Error F/F circuits are used for parity error detection and set the corresponding status bit.

The Framing Error Flag F/F is set if the Stop bit is absent at the end of the data byte (asynchronous mode), and also sets the corresponding status bit.

RxRDY (Receiver Ready)

This output indicates that the 8251A contains a character that is ready to be input to the CPU. Rx RDY can be connected to the interrupt structure of the CPU or, for Polled operation, the CPU can check the condition of RxRDY using a Status Read operation.

Rx Enable off both masks and holds RxRDY in the Reset Condition. For Asynchronous mode, to set RxRDY, the Receiver must be Enabled to sense a Start Bit and a complete character must be assembled and transferred to the Data Output Register. For Synchronous mode, to set RxRDY, the Receiver must be enabled and a character must finish assembly and be transferred to the Data Output Register.

Failure to read the received character from the Rx Data Output Register prior to the assembly of the next Rx Data character will set overrun condition error and the previous character will be written over and lost. If the Rx Data is being read by the CPU when the internal transfer is occurring, overrun error will be set and the old character will be lost.

$\overline{RxC}$ (Receiver Clock)

The Receiver Clock controls the rate at which the character is to be received. In Synchronous Mode, the Baud Rate (1x) is equal to the actual frequency of $\overline{RxC}$. In Asynchronous Mode, the Baud Rate is a fraction of the actual $\overline{RxC}$ frequency. A portion of the mode instruction selects this factor; 1, 1/16 or 1/64 the $\overline{RxC}$.

For Example:

Baud Rate equals 300 Baud, if
$\overline{RxC}$ equals 300 Hz (1x)
$\overline{RxC}$ equals 4800 Hz (16x)
$\overline{RxC}$ equals 19.2 kHz (64x).

Baud Rate equals 2400 Baud, if
$\overline{RxC}$ equals 2400 Hz (1x)
$\overline{RxC}$ equals 38.4 kHz (16x)
$\overline{RxC}$ equals 153.6 kHz (64x).

Data is sampled into the 8251A on the rising edge of $\overline{RxC}$.

NOTE: In most communications systems, the 8251A will be handling both the transmission and reception operations of a single link. Consequently, the Receive and Transmit Baud Rates will be the same. Both $\overline{TxC}$ and $\overline{RxC}$ will require identical frequencies for this operation and can be tied together and connected to a single frequency source (Baud Rate Generator) to simplify the interface.

SYNDET (SYNC Detect)/BRKDET (Break Detect)

This pin is used in SYNChronous Mode for SYNDET and may be used as either input or output, programmable through the Control Word. It is reset to output mode low upon RESET. When used as an output (internal Sync mode), the SYNDET pin will go "high" to indicate that the 8251A has located the SYNC character in the Receive mode. If the 8251A is programmed to use double Sync characters (bisync), then SYNDET will go "high" in the middle of the last bit of the second Sync character. SYNDET is automatically reset upon a Status Read operation.

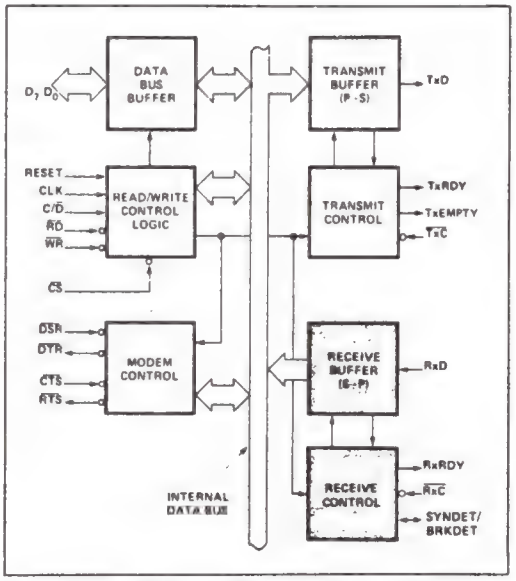


Figure 5. 8251A Block Diagram Showing Receiver Buffer and Control Functions

When used as an input (external SYNC detect mode), a positive going signal will cause the 8251A to start assembling data characters on the rising edge of the next $\overline{RxC}$. Once in SYNC, the "high" input signal can be removed. When External SYNC Detect is programmed, the Internal SYNC Detect is disabled.

BREAK DETECT (Async Mode Only)

This output will go high whenever the receiver remains low through two consecutive stop bit sequences (including the start bits, data bits, and parity bits). Break Detect may also be read as a Status bit. It is reset only upon a master chip Reset or Rx Data returning to a "one" state.

NOTE: On the 8251A/S2657, if the RxData returns to a "one" state during the last bit of the next character after the break, break detect will latch-up, and the device must be cleared by a Chip Reset.

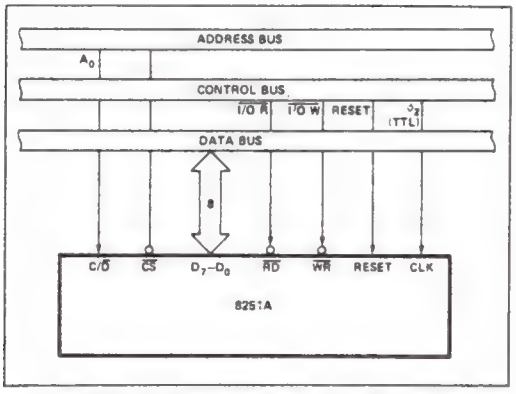


Figure 6. 8251A Interface to 8080 Standard System Bus

DETAILED OPERATION DESCRIPTION

General

The complete functional definition of the 8251A is programmed by the system's software. A set of control words must be sent out by the CPU to initialize the 8251A to support the desired communications format. These control words will program the: BAUD RATE, CHARACTER LENGTH, NUMBER OF STOP BITS, SYNCHRONOUS or ASYNCHRONOUS OPERATION, EVEN/ODD/OFF PARITY, etc. In the Synchronous Mode, options are also provided to select either internal or external character synchronization.

Once programmed, the 8251A is ready to perform its communication functions. The TxRDY output is raised "high" to signal the CPU that the 8251A is ready to receive a data character from the CPU. This output (TxRDY) is reset automatically when the CPU writes a character into the 8251A. On the other hand, the 8251A receives serial data from the MODEM or I/O device. Upon receiving an entire character, the RxRDY output is raised "high" to signal the CPU that the 8251A has a complete character ready for the CPU to fetch. RxRDY is reset automatically upon the CPU data read operation.

The 8251A cannot begin transmission until the Tx Enable (Transmitter Enable) bit is set in the Command Instruction and it has received a Clear To Send ($\overline{CTS}$) input. The TxD output will be held in the marking state upon Reset.

Programming the 8251A

Prior to starting data transmission or reception, the 8251A must be loaded with a set of control words generated by the CPU. These control signals define the complete functional definition of the 8251A and must immediately follow a Reset operation (internal or external).

The control words are split into two formats:

1. Mode Instruction
2. Command Instruction

Mode Instruction

This format defines the general operational characteristics of the 8251A. It must follow a Reset operation (internal or external). Once the Mode Instruction has been written into the 8251A by the CPU, SYNC characters or Command Instructions may be inserted.

Command Instruction

This format defines a status word that is used to control the actual operation of the 8251A.

Both the Mode and Command Instructions must conform to a specified sequence for proper device operation. The Mode Instruction must be inserted immediately following a Reset operation, prior to using the 8251A for data communication.

All control words written into the 8251A after the Mode Instruction will load the Command Instruction. Command Instructions can be written into the 8251A at any time in the data block during the operation of the 8251A. To return to the Mode Instruction format, the master Reset bit in the Command Instruction word can be set to initiate an internal Reset operation which automatically places the 8251A back into the Mode Instruction format. Command Instructions must follow the Mode Instructions or Sync characters.

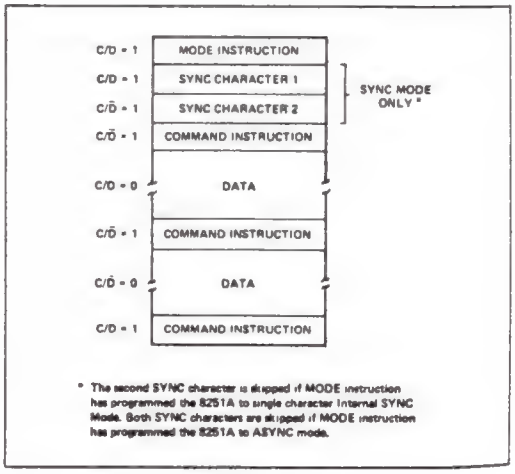

C/D = 1
MODE INSTRUCTION
C/D = 1
SYNC CHARACTER 1
C/D = 1
SYNC CHARACTER 2
SYNC MODE ONLY *
C/D = 1
COMMAND INSTRUCTION
C/D = 0
DATA
C/D = 1
COMMAND INSTRUCTION
C/D = 0
DATA
C/D = 1
COMMAND INSTRUCTION
* The second SYNC character is skipped if MODE instruction has programmed the 8251A to single character Internal SYNC Mode. Both SYNC characters are skipped if MODE instruction has programmed the 8251A to ASYNC mode.

Figure 7. Typical Data Block

AFN-01573B

Mode Instruction Definition

The 8251A can be used for either Asynchronous or Synchronous data communication. To understand how the Mode Instruction defines the functional operation of the 8251A, the designer can best view the device as two separate components sharing the same package, one Asynchronous the other Synchronous. The format definition can be changed only after a master chip Reset. For explanation purposes the two formats will be isolated.

NOTE: When parity is enabled it is not considered as one of the data bits for the purpose of programming the word length. The actual parity bit received on the Rx Data line cannot be read on the Data Bus. In the case of a programmed character length of less than 8 bits, the least significant Data Bus bits will hold the data; unused bits are "don't care" when writing data to the 8251A, and will be "zeros" when reading the data from the 8251A.

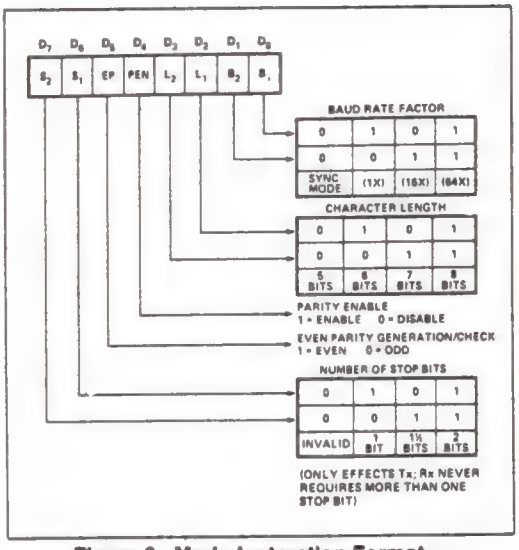


Figure 8. Mode Instruction Format, Asynchronous Mode

Asynchronous Mode (Transmission)

Whenever a data character is sent by the CPU the 8251A automatically adds a Start bit (low level) followed by the data bits (least significant bit first), and the programmed number of Stop bits to each character. Also, an even or odd Parity bit is inserted prior to the Stop bit(s), as defined by the Mode Instruction. The character is then transmitted as a serial data stream on the TxD output. The serial data is shifted out on the falling edge of $\overline{TxC}$ at a rate equal to 1, 1/16, or 1/64 that of the $\overline{TxC}$, as defined by the Mode Instruction. BREAK characters can be continuously sent to the TxD if commanded to do so.

When no data characters have been loaded into the 8251A the TxD output remains "high" (marking) unless a Break (continuously low) has been programmed.

Asynchronous Mode (Receive)

The RxD line is normally high. A falling edge on this line triggers the beginning of a START bit. The validity of this START bit is checked by again strobing this bit at its nominal center (16X or 64X mode only). If a low is detected again, it is a valid START bit, and the bit counter will start counting. The bit counter thus locates the center of the data bits, the parity bit (if it exists) and the stop bits. If parity error occurs, the parity error flag is set. Data and parity bits are sampled on the RxD pin with the rising edge of $\overline{RxC}$. If a low level is detected as the STOP bit, the Framing Error flag will be set. The STOP bit signals the end of a character. Note that the *receiver* requires only *one* stop bit, regardless of the number of stop bits programmed. This character is then loaded into the parallel I/O buffer of the 8251A. The RxRDY pin is raised to signal the CPU that a character is ready to be fetched. If a previous character has not been fetched by the CPU, the present character replaces it in the I/O buffer, and the OVERRUN Error flag is raised (thus the previous character is lost). All of the error flags can be reset by an Error Reset Instruction. The occurrence of any of these errors will not affect the operation of the 8251A.

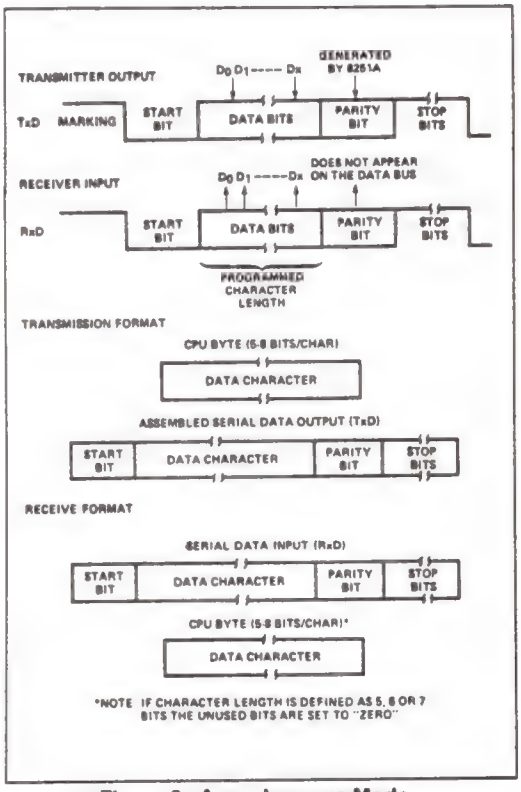


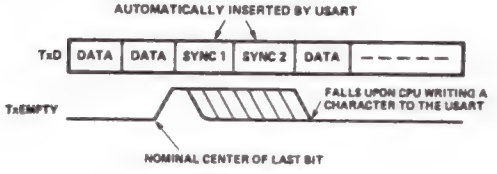
Figure 9. Asynchronous Mode

AFN-015738

Synchronous Mode (Transmission)

The TxD output is continuously high until the CPU sends its first character to the 8251A which usually is a SYNC character. When the $\overline{CTS}$ line goes low, the first character is serially transmitted out. All characters are shifted out on the falling edge of $\overline{TxC}$. Data is shifted out at the same rate as the $\overline{TxC}$.

Once transmission has started, the data stream at the TxD output must continue at the $\overline{TxC}$ rate. If the CPU does not provide the 8251A with a data character before the 8251A Transmitter Buffers become empty, the SYNC characters (or character if in single SYNC character mode) will be automatically inserted in the TxD data stream. In this case, the TxEMPTY pin is raised high to signal that the 8251A is empty and SYNC characters are being sent out. TxEMPTY does not go low when the SYNC is being shifted out (see figure below). The TxEMPTY pin is internally reset by a data character being written into the 8251A.



Synchronous Mode (Receive)

In this mode, character synchronization can be internally or externally achieved. If the SYNC mode has been programmed, ENTER HUNT command should be included in the first command instruction word written. Data on the RxD pin is then sampled in on the rising edge of $\overline{RxC}$. The content of the Rx buffer is compared at every bit boundary with the first SYNC character until a match occurs. If the 8251A has been programmed for two SYNC characters, the subsequent received character is also compared; when both SYNC characters have been detected, the USART ends the HUNT mode and is in character synchronization. The SYNDET pin is then set high, and is reset automatically by a STATUS READ. If parity is programmed, SYNDET will not be set until the middle of the parity bit instead of the middle of the last data bit.

In the external SYNC mode, synchronization is achieved by applying a high level on the SYNDET pin, thus forcing the 8251A out of the HUNT mode. The high level can be removed after one $\overline{RxC}$ cycle. An ENTER HUNT command has no effect in the asynchronous mode of operation.

Parity error and overrun error are both checked in the same way as in the Asynchronous Rx mode. Parity is checked when not in Hunt, regardless of whether the Receiver is enabled or not.

The CPU can command the receiver to enter the HUNT mode if synchronization is lost. This will also set all the used character bits in the buffer to a "one", thus preventing a possible false SYNDET caused by data that happens to be in the Rx Buffer at ENTER HUNT time. Note that the SYNDET F/F is reset at each Status Read, regardless of whether internal or external SYNC has been programmed. This does not cause the 8251A to return to the HUNT mode. When in SYNC mode, but not in HUNT, Sync Detection is still functional, but only occurs at the "known" word boundaries. Thus, if one Status Read indicates SYNDET and a second Status Read also indicates SYNDET, then the programmed SYNDET characters have been received since the previous Status Read. (If double character sync has been programmed, then both sync characters have been contiguously received to gate a SYNDET indication.) When external SYNDET mode is selected, internal Sync Detect is disabled, and the SYNDET F/F may be set at any bit boundary.

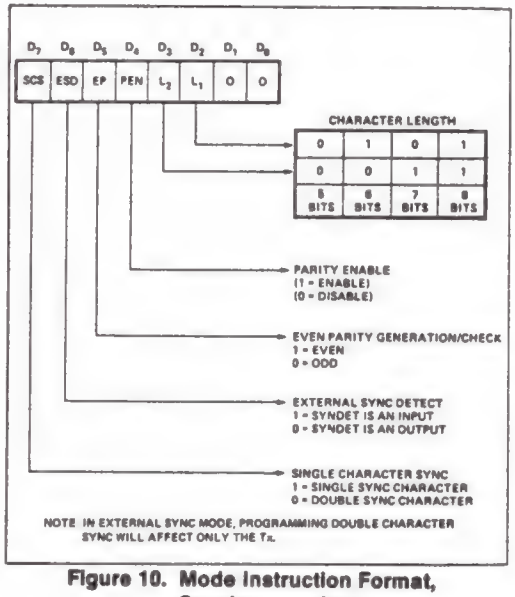

D7 D6 D5 D4 D3 D2 D1 D0
SCS ESD EP PEN L2 L1 0 0
CHARACTER LENGTH
0 1 0 1
0 0 1 1
5 BITS 6 BITS 7 BITS 8 BITS
PARITY ENABLE
(1 = ENABLE)
(0 = DISABLE)
EVEN PARITY GENERATION/CHECK
1 = EVEN
0 = ODD
EXTERNAL SYNC DETECT
1 = SYNDET IS AN INPUT
0 = SYNDET IS AN OUTPUT
SINGLE CHARACTER SYNC
1 = SINGLE SYNC CHARACTER
0 = DOUBLE SYNC CHARACTER
NOTE: IN EXTERNAL SYNC MODE, PROGRAMMING DOUBLE CHARACTER SYNC WILL AFFECT ONLY THE Tx.

Figure 10. Mode Instruction Format, Synchronous Mode

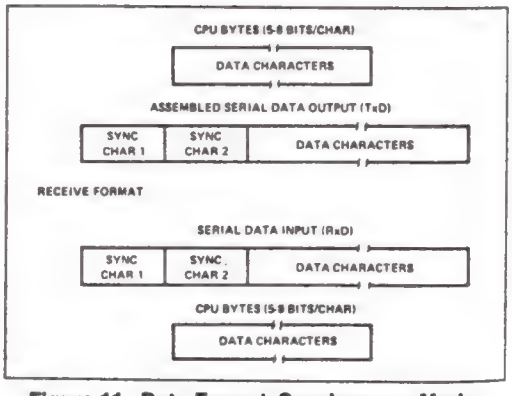


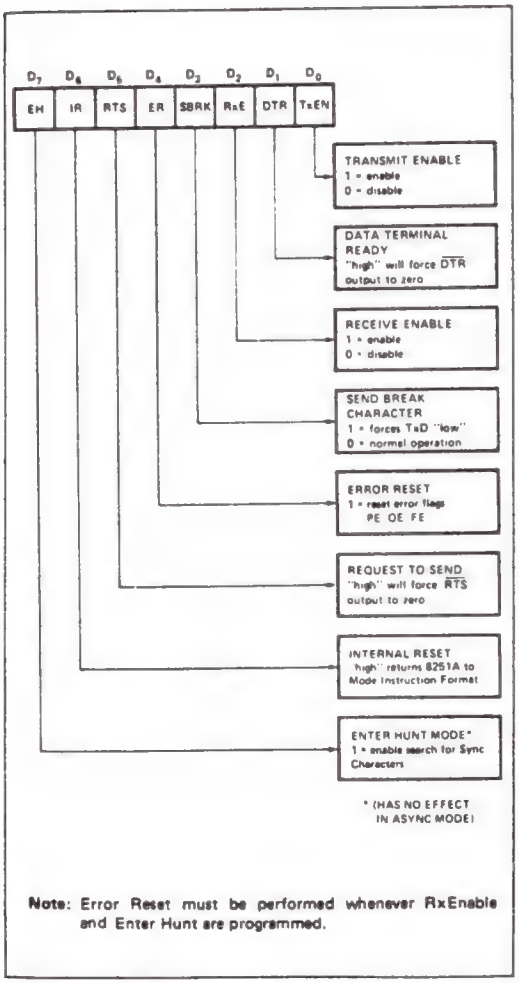
Figure 11. Data Format, Synchronous Mode

AFN-01573B

COMMAND INSTRUCTION DEFINITION

Once the functional definition of the 8251A has been programmed by the Mode Instruction and the Sync Characters are loaded (if in Sync Mode) then the device is ready to be used for data communication. The Command Instruction controls the actual operation of the selected format. Functions such as: Enable Transmit/Receive, Error Reset and Modem Controls are provided by the Command Instruction.

Once the Mode Instruction has been written into the 8251A and Sync characters inserted, if necessary, then all further "control writes" ($C/\overline{D} = 1$) will load a Command Instruction. A Reset Operation (internal or external) will return the 8251A to the Mode Instruction format.


D7 D6 D5 D4 D3 D2 D1 D0
EH IR RTS ER SBRK RxE DTR TxEN
TRANSMIT ENABLE
1 = enable
0 = disable
DATA TERMINAL READY
"high" will force DTR output to zero
RECEIVE ENABLE
1 = enable
0 = disable
SEND BREAK CHARACTER
1 = forces TxD "low"
0 = normal operation
ERROR RESET
1 = reset error flags PE OE FE
REQUEST TO SEND
"high" will force RTS output to zero
INTERNAL RESET
"high" returns 8251A to Mode Instruction Format
ENTER HUNT MODE*
1 = enable search for Sync Characters
* (HAS NO EFFECT IN ASYNC MODE)

Note: Error Reset must be performed whenever RxEnable and Enter Hunt are programmed.

Figure 12. Command Instruction Format

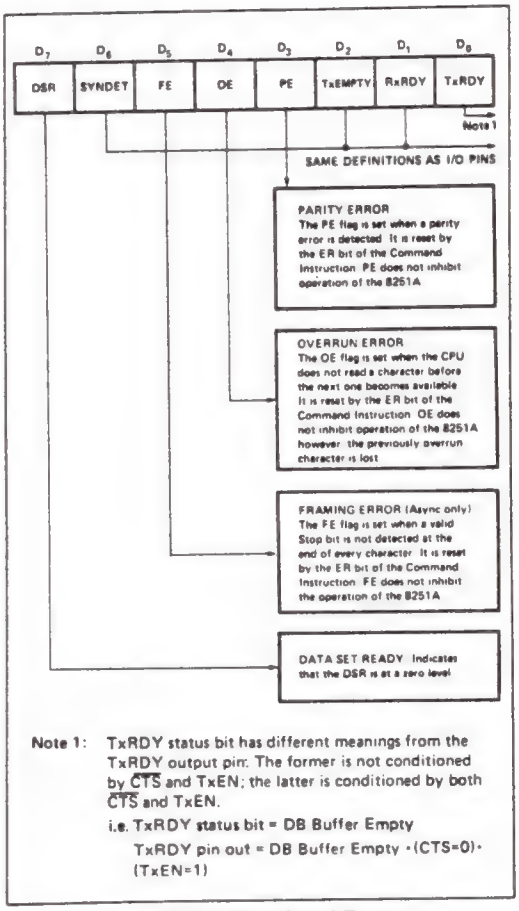
STATUS READ DEFINITION

In data communication systems it is often necessary to examine the "status" of the active device to ascertain if errors have occurred or other conditions that require the processor's attention. The 8251A has facilities that allow the programmer to "read" the status of the device at any time during the functional operation. (The status update is inhibited during status read).

A normal "read" command is issued by the CPU with $C/\overline{D} = 1$ to accomplish this function.

Some of the bits in the Status Read Format have identical meanings to external output pins so that the 8251A can be used in a completely Polled environment or in an interrupt driven environment. TxRDY is an exception.

Note that status update can have a maximum delay of 28 clock periods from the actual event affecting the status.


D7 D6 D5 D4 D3 D2 D1 D0
DSR SYNDET FE OE PE TxEMPTY RxRDY TxRDY
Note 1
SAME DEFINITIONS AS I/O PINS
PARITY ERROR
The PE flag is set when a parity error is detected. It is reset by the ER bit of the Command Instruction. PE does not inhibit operation of the 8251A
OVERRUN ERROR
The OE flag is set when the CPU does not read a character before the next one becomes available. It is reset by the ER bit of the Command Instruction. OE does not inhibit operation of the 8251A however the previously overrun character is lost
FRAMING ERROR (Async only)
The FE flag is set when a valid Stop bit is not detected at the end of every character. It is reset by the ER bit of the Command Instruction. FE does not inhibit the operation of the 8251A
DATA SET READY. Indicates that the DSR is at a zero level

Note 1: TxRDY status bit has different meanings from the TxRDY output pin. The former is not conditioned by $\overline{CTS}$ and TxEN; the latter is conditioned by both $\overline{CTS}$ and TxEN.

i.e. TxRDY status bit = DB Buffer Empty

TxRDY pin out = DB Buffer Empty · (CTS=0) · (TxEN=1)

Figure 13. Status Read Format

APPLICATIONS OF THE 8251A

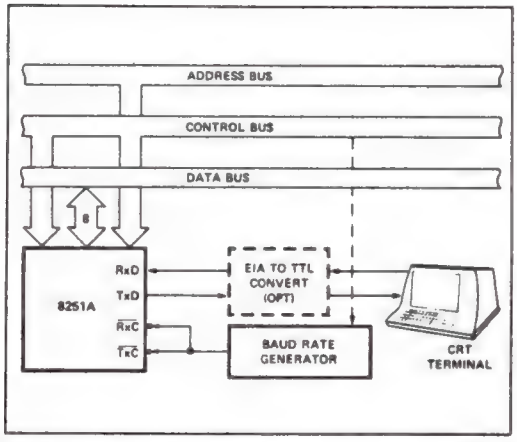


Figure 14. Asynchronous Serial Interface to CRT Terminal, DC—9600 Baud

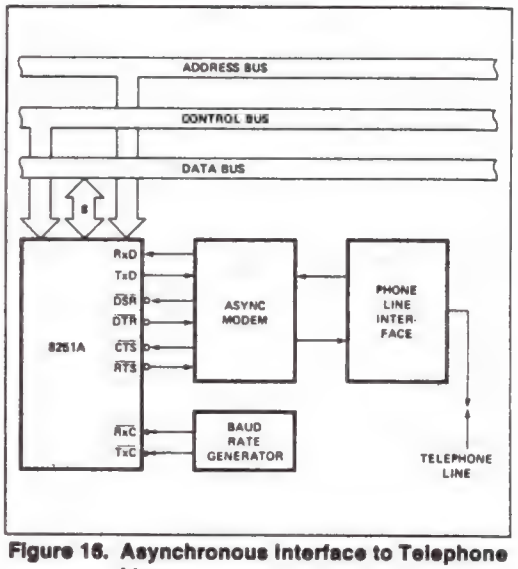


Figure 16. Asynchronous Interface to Telephone Lines

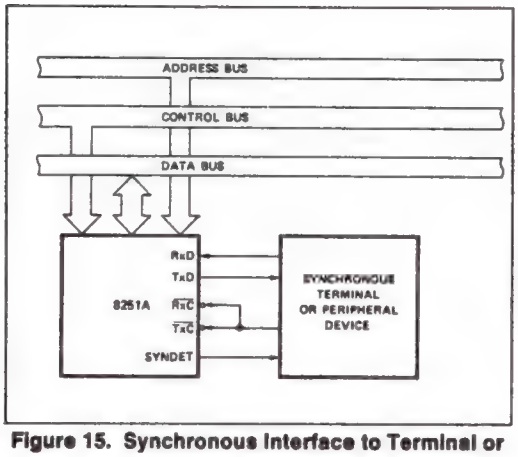


Figure 15. Synchronous Interface to Terminal or Peripheral Device

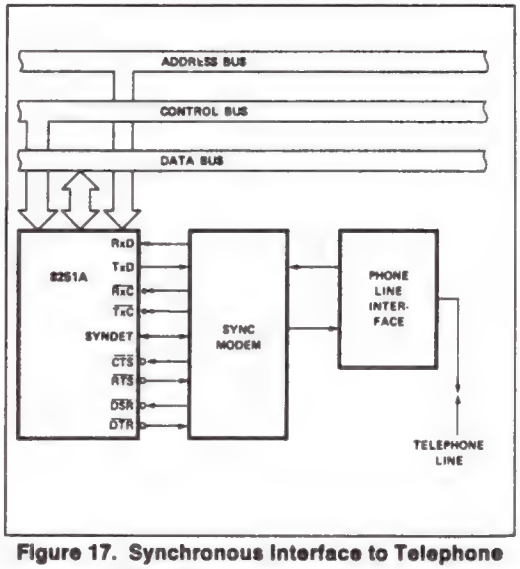


Figure 17. Synchronous Interface to Telephone Lines

AFN-01573B

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias. . . . . 0°C to 70°C
Storage Temperature . . . . . –65°C to +150°C
Voltage On Any Pin
With Respect to Ground . . . . . –0.5V to +7V
Power Dissipation . . . . . 1 Watt

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5.0V ±5%, GND = 0V)

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage	–0.5	0.8	V	
V_{IH}	Input High Voltage	2.2	V_{CC}	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.2 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = –400 μA
I_{OFL}	Output Float Leakage		±10	μA	V_{OUT} = V_{CC} TO 0.45V
I_{IL}	Input Leakage		±10	μA	V_{IN} = V_{CC} TO 0.45V
I_{CC}	Power Supply Current		100	mA	All Outputs = High

CAPACITANCE (T_A = 25°C, V_{CC} = GND = 0V)

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
C_{IN}	Input Capacitance		10	pF	fc = 1MHz
$C_{I/O}$	I/O Capacitance		20	pF	Unmeasured pins returned to GND

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5.0V ±5%, GND = 0V)

Bus Parameters (Note 1)

READ CYCLE

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{AR}	Address Stable Before $\overline{READ}$ ($\overline{CS}$, C/$\overline{D}$)	50		ns	Note 2
t_{RA}	Address Hold Time for $\overline{READ}$ ($\overline{CS}$, C/$\overline{D}$)	50		ns	Note 2
t_{RR}	$\overline{READ}$ Pulse Width	250		ns	
t_{RD}	Data Delay from $\overline{READ}$		250	ns	3, C_L = 150 pF
t_{DF}	$\overline{READ}$ to Data Floating	10	100	ns	

AFN-01573B

A.C. CHARACTERISTICS (Continued)

WRITE CYCLE

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{AW}	Address Stable Before $\overline{WRITE}$	50		ns	
t_{WA}	Address Hold Time for $\overline{WRITE}$	50		ns	
t_{WW}	$\overline{WRITE}$ Pulse Width	250		ns	
t_{DW}	Data Set Up Time for $\overline{WRITE}$	150		ns	
t_{WD}	Data Hold Time for $\overline{WRITE}$	50		ns	
t_{RV}	Recovery Time Between WRITES	6		t_{CY}	Note 4

OTHER TIMINGS

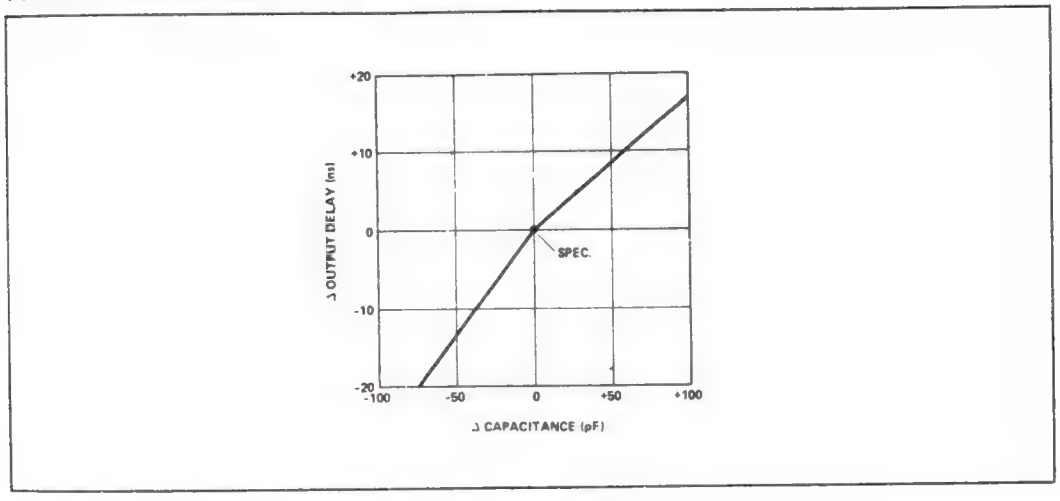
Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{CY}	Clock Period	320	1350	ns	Notes 5, 6
t_{ϕ}	Clock High Pulse Width	140	t_{CY-90}	ns	
$t_{\overline{\phi}}$	Clock Low Pulse Width	90		ns	
t_R, t_F	Clock Rise and Fall Time		20	ns	
t_{DTx}	TxD Delay from Falling Edge of $\overline{TxC}$		1	μs	
f_{Tx}	Transmitter Input Clock Frequency				
	1x Baud Rate	DC	64	kHz	
	16x Baud Rate	DC	310	kHz	
	64x Baud Rate	DC	615	kHz	
t_{TPW}	Transmitter Input Clock Pulse Width				
	1x Baud Rate	12		t_{CY}	
	16x and 64x Baud Rate	1		t_{CY}	
t_{TPD}	Transmitter Input Clock Pulse Delay				
	1x Baud Rate	15		t_{CY}	
	16x and 64x Baud Rate	3		t_{CY}	
f_{Rx}	Receiver Input Clock Frequency				
	1x Baud Rate	DC	64	kHz	
	16x Baud Rate	DC	310	kHz	
	64x Baud Rate	DC	615	kHz	
t_{RPW}	Receiver Input Clock Pulse Width				
	1x Baud Rate	12		t_{CY}	
	16x and 64x Baud Rate	1		t_{CY}	
t_{RPD}	Receiver Input Clock Pulse Delay				
	1x Baud Rate	15		t_{CY}	
	16x and 64x Baud Rate	3		t_{CY}	
t_{TxRDY}	TxRDY Pin Delay from Center of last Bit		8	t_{CY}	Note 7
$t_{TxRDY\ CLEAR}$	TxRDY ↓ from Leading Edge of $\overline{WR}$		6	t_{CY}	Note 7
t_{RxRDY}	RxRDY Pin Delay from Center of last Bit		24	t_{CY}	Note 7
$t_{RxRDY\ CLEAR}$	RxRDY ↓ from Leading Edge of $\overline{RD}$		6	t_{CY}	Note 7
t_{IS}	Internal SYNDET Delay from Rising Edge of $\overline{RxC}$		24	t_{CY}	Note 7
t_{ES}	External SYNDET Set-Up Time Before Falling Edge of $\overline{RxC}$	16		t_{CY}	Note 7
$t_{TxEMPTY}$	TxEMPTY Delay from Center of Last Bit	20		t_{CY}	Note 7
t_{WC}	Control Delay from Rising Edge of WRITE (TxEn, $\overline{DTR}$, $\overline{RTS}$)	8		t_{CY}	Note 7
t_{CR}	Control to READ Set-Up Time ($\overline{DSR}$, $\overline{CTS}$)	20		t_{CY}	Note 7

AFN-01573B

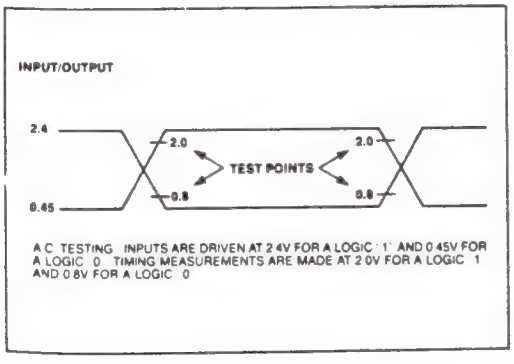
A.C. CHARACTERISTICS (Continued)

NOTES:
1. AC timings measured V_{OH} = 2.0, V_{OL} = 0.8, and with load circuit of Figure 1.
2. Chip Select (CS) and Command/Data (C/D) are considered as Addresses.
3. Assumes that Address is valid before $R_D\downarrow$.
4. This recovery time is for Mode Initialization only. Write Data is allowed only when TxRDY = 1. Recovery Time between Writes for Asynchronous Mode is 8 t_{CY} and for Synchronous Mode is 16 t_{CY}.
5. The TxC and RxC frequencies have the following limitations with respect to CLK: For 1x Baud Rate, f_{Tx} or $f_{Rx} \leq 1/(30\ t_{CY})$; For 16x and 64x Baud Rate, f_{Tx} or $f_{Rx} \leq 1/(4.5\ t_{CY})$.
6. Reset Pulse Width = 6 t_{CY} minimum; System Clock must be running during Reset.
7. Status update can have a maximum delay of 28 clock periods from the event affecting the status.

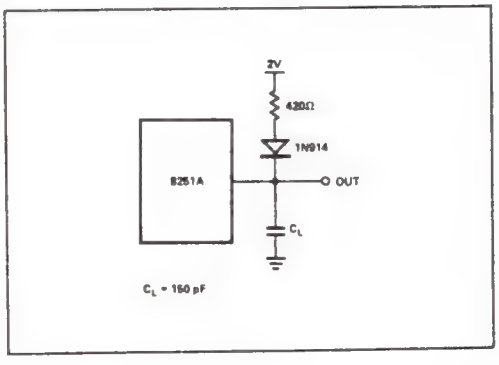
TYPICAL Δ OUTPUT DELAY VS. Δ CAPACITANCE (pF)



A.C. TESTING INPUT, OUTPUT WAVEFORM


INPUT/OUTPUT
2.4
0.45
2.0
0.8
TEST POINTS
2.0
0.8
A.C. TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0". TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC "1" AND 0.8V FOR A LOGIC "0".

A.C. TESTING LOAD CIRCUIT

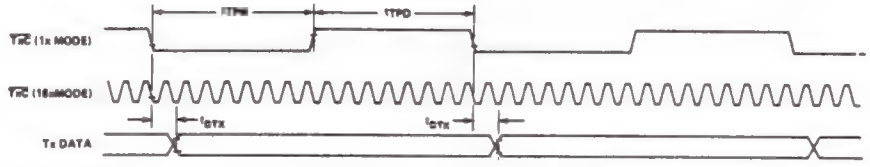


AFN-015738

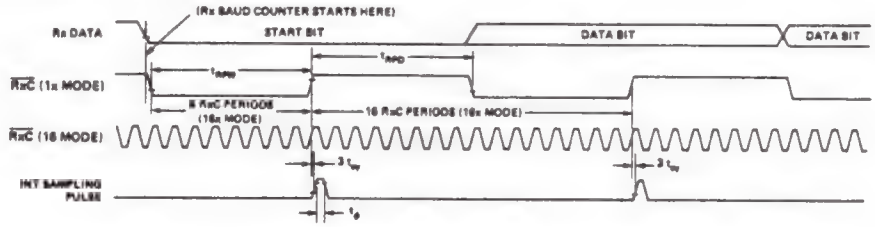
WAVEFORMS

SYSTEM CLOCK INPUT

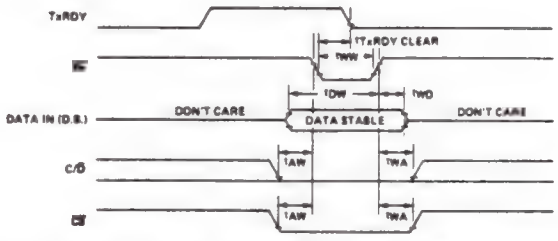
TRANSMITTER CLOCK AND DATA



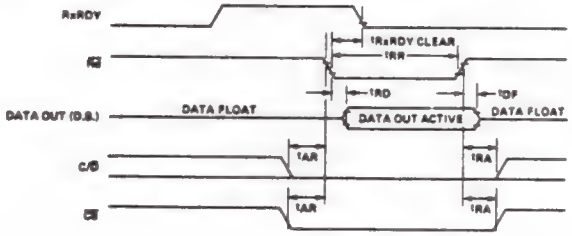
RECEIVER CLOCK AND DATA



WRITE DATA CYCLE (CPU → USART)



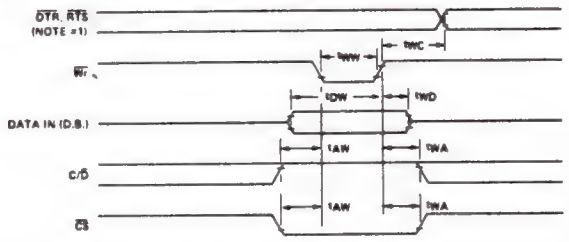
READ DATA CYCLE (CPU ← USART)



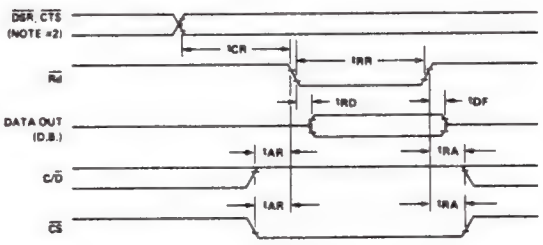
AFN-01573B

WAVEFORMS (Continued)

WRITE CONTROL OR OUTPUT PORT CYCLE (CPU → USART)

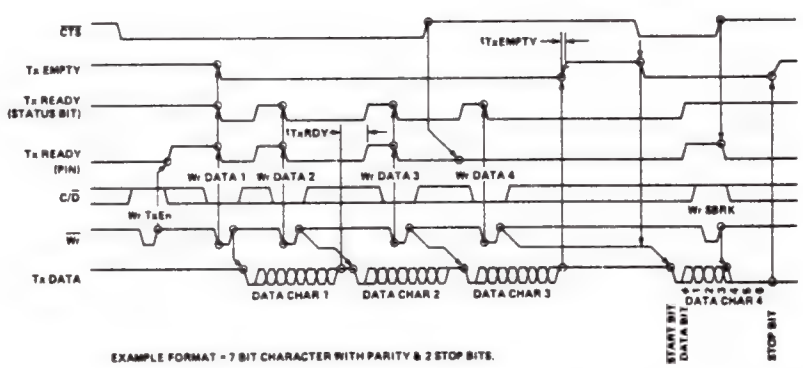


READ CONTROL OR INPUT PORT (CPU ← USART)



NOTE #1: T_{WC} INCLUDES THE RESPONSE TIMING OF A CONTSOL BYTE.
NOTE #2: T_{CR} INCLUDES THE EFFECT OF CTS ON THE TxENBL CIRCUITRY.

TRANSMITTER CONTROL AND FLAG TIMING (ASYNC MODE)

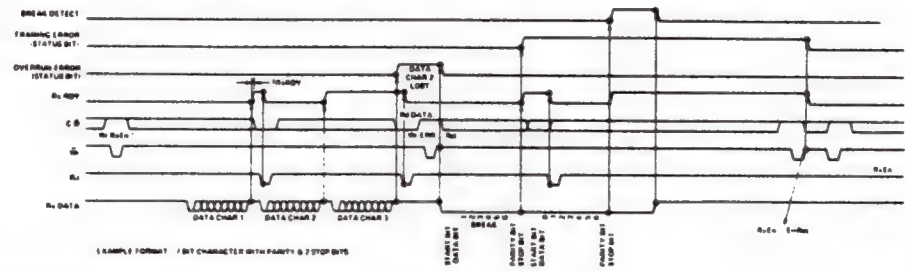


EXAMPLE FORMAT = 7 BIT CHARACTER WITH PARITY & 2 STOP BITS.

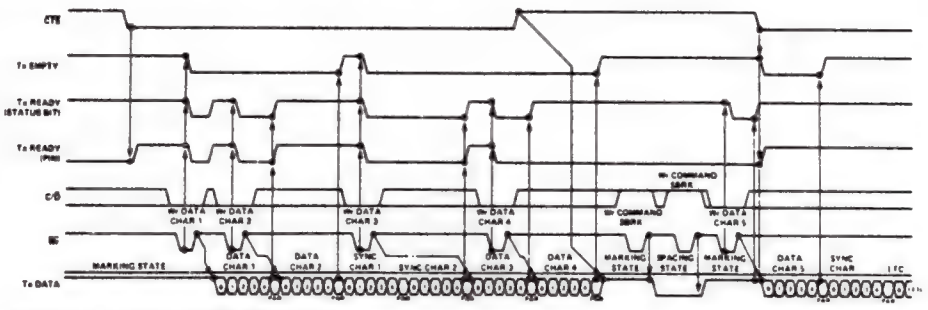
AFN-01573B

WAVEFORMS (Continued)

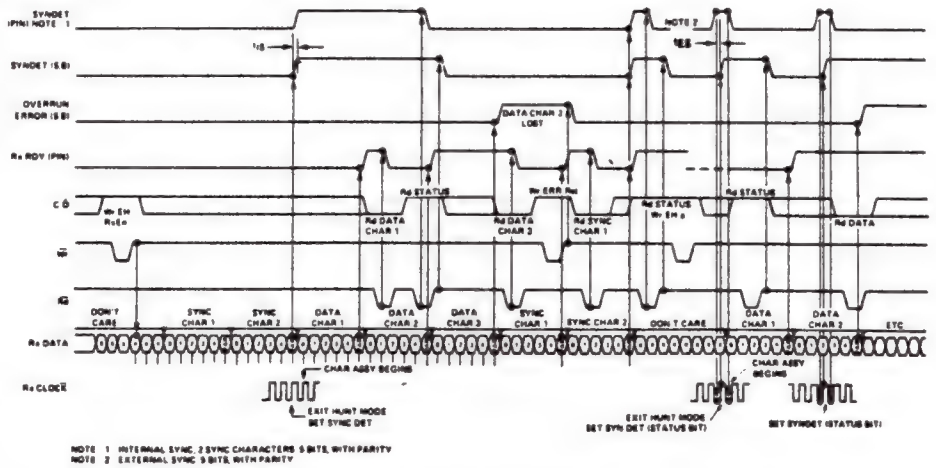
RECEIVER CONTROL AND FLAG TIMING (ASYNC MODE)



TRANSMITTER CONTROL AND FLAG TIMING (SYNC MODE)



RECEIVER CONTROL AND FLAG TIMING (SYNC MODE)



AFN-01573B

Master Reset Feature

A master reset of all computer hardware may be accomplished by depressing the solid colored RED keys located on either side of the alphanumeric keyboard.

Cursor Control Keys

There are four cursor control keys located on every CompuStar VPU. These keys are located on the right side of the numeric keypad. These keys will transmit codes to any program running on the VPU. These codes may in turn be interpreted by the program to result in cursor movement on the screen. It is important to know that these keys will not produce cursor movement when you are in the operating system mode. The reason for this is that CP/M does not define any use of cursor positioning on the screen. As such, depression of these keys while in the operating system mode will result in the control codes assigned to the individual keys being displayed as control codes on the screen.

Accessing Time/Date Data

Accessing the TIME/DATE data is accomplished by reading the appropriate port (31H through 3CH as specified in the Table of I/O Ports in this section). If the real time clock is being updated when the read is attempted, the low order four bits returned will be 1111, indicating a hexadecimal F. The read must be retried if this occurs until a correct value is returned. The subroutine program that follows illustrates one way to do this. It is written in MBASIC.

```
2000 REM CompuStar Time of Day Routine
2010 REM
2020 REM This subroutine returns the time-of-day which is currently set in the CompuStar TOD
2030 REM clock. The time is returned in the variable T$. It is a string of length 10 where the
2040 REM format is HH:MM:SS:T.
2050 REM
2060 T$ = " "
2070 FOR I = 6 to 0 STEP -1
2080 V = (INP(&H31 + I) AND &HF)
2090 IF V = 15 THEN 2080
2100 T$ = T$ + MID$ (STR$(V),2)
2110 IF I MOD 2 = 1 THEN T$ = T$ + ":"
2115 NEXT I
2120 RETURN
```

INTERFACING INFORMATION

RS232C Serial Interface

The following chart illustrates the pinouts for the MAIN and AUXILIARY serial ports and the direction of signal flow.

COMPUSTAR SERIAL PORT PIN ASSIGNMENTS

MAIN PORT

PIN #	ASSIGNMENT	DIRECTION
1	GND	–
2	TRANSMITTED DATA	(FROM VPU)
3	RECEIVED DATA	(TO VPU)
4	REQUEST TO SEND	(FROM VPU)
5*	CLEAR TO SEND	(TO VPU)
6	DATA SET READY	(TO VPU)
7	GND	–
15	TRANSMIT CLOCK	(TO VPU)
17	RECEIVE CLOCK	(TO VPU)
20	DATA TERMINAL READY	(FROM VPU)
22	RING INDICATOR	(TO VPU)
24	CLOCK	(FROM VPU)

*Pin 5 must be at a high level at the connector in order for transmission.

AUXILIARY PORT

PIN #	ASSIGNMENT	DIRECTION
1	GND	–
2	RECEIVED DATA	(TO VPU)
3	TRANSMITTED DATA	(FROM VPU)
7	GND	–
20	DATA TERMINAL READY	(TO VPU)

Bus Adaptor Interface

The CompuStar VPU contains a Z80 bus interface to the main processor bus. These signals occupy the lower 34 pins of a 50 pin connector (J1) on the processor board. A chart defining all the signals on this connector is shown on the following pages.

When using this interface, it is recommended that all signals be buffered so as not to excessively load the main processor bus. The external bus should **ONLY** be utilized for **I/O** devices using addresses 80 to FFH. Memory mapped I/O is **NOT** possible for user applications since the VPU is internally configured for 64K of RAM.

PIN CONNECTIONS FOR EXTERNAL BUS

PIN NO.	SIGNAL NAME	INPUT OR OUTPUT	DESCRIPTION
1	OUT*	OUTPUT	PERIPHERAL WRITE STROBE OUTPUT
2	A11	OUTPUT	ADDRESS OUTPUT
3	WR*	OUTPUT	MEMORY WRITE STROBE OUTPUT
4	A14	OUTPUT	ADDRESS OUTPUT
5	RD*	OUTPUT	MEMORY READ STROBE OUTPUT
6	D4	BOTH	BIDIRECTIONAL DATA BUS
7	IN*	OUTPUT	PERIPHERAL READ STROBE OUTPUT
8	D7	BOTH	BIDIRECTIONAL DATA BUS
9	GND	N/A	SIGNAL GROUND
10	N/A	N/A	N/A
11	A10	OUTPUT	ADDRESS OUTPUT
12	SYSRES	OUTPUT	SYSTEM RESET OUTPUT, LOW DURING POWER UP INITIALIZE OR RESET DEPRESSED
13	A0	OUTPUT	ADDRESS OUTPUT
14	D6	BOTH	BIDIRECTIONAL DATA BUS
15	A12	OUTPUT	ADDRESS OUTPUT
16	A13	OUTPUT	ADDRESS OUTPUT
17	A15	OUTPUT	ADDRESS OUTPUT
18	D3	BOTH	BIDIRECTIONAL DATA BUS
19	D5	BOTH	BIDIRECTIONAL DATA BUS
20	D0	BOTH	BIDIRECTIONAL DATA BUS
21	A8	OUTPUT	ADDRESS OUTPUT
22	A4	OUTPUT	ADDRESS OUTPUT
23	D2	BOTH	BIDIRECTIONAL DATA BUS
24	A1	OUTPUT	ADDRESS OUTPUT
25	A3	OUTPUT	ADDRESS OUTPUT
26	A5	OUTPUT	ADDRESS OUTPUT
27	A9	OUTPUT	ADDRESS OUTPUT
28	A7	OUTPUT	ADDRESS OUTPUT
29	A2	OUTPUT	ADDRESS OUTPUT
30	A6	OUTPUT	ADDRESS OUTPUT
31	D1	BOTH	BIDIRECTIONAL DATA BUS
32	+5V	N/A	POSITIVE 5 VOLTS (LIMITED CURRENT)
33	GND	N/A	SIGNAL GROUND
34	GND	N/A	SIGNAL GROUND
35	GND	N/A	SIGNAL GROUND
36	+12V	N/A	POSITIVE 12 VOLTS (used for RS232 Receiver bias)
37	AUX RX DATA	INPUT	AUXILIARY PORT RECEIVE DATA
38	MAIN TX CLK	INPUT	MAIN PORT TRANSMIT CLOCK
39	MAIN RX CLK	INPUT	MAIN PORT RECEIVE CLOCK
40	MAIN RX DATA	INPUT	MAIN PORT RECEIVE DATA
41	MAIN CTS	INPUT	MAIN PORT CLEAR TO SEND
42	AUX DSR	INPUT	AUXILIARY PORT DATA SET READY
43	MAIN RTS	OUTPUT	MAIN PORT REQUEST TO SEND
44	MAIN DSR	INPUT	MAIN PORT DATA SET READY
45	MAIN CLK	OUTPUT	MAIN PORT CLOCK
46	AUX TX DATA	OUTPUT	AUXILIARY PORT TRANSMIT DATA
47	MAIN RI	INPUT	MAIN PORT RING INDICATOR
48	−12V	N/A	MINUS 12 VOLTS (used for RS232 Receiver bias)
49	MAIN DTR	OUTPUT	MAIN PORT DATA TERMINAL READY
50	MAIN TX DATA	OUTPUT	MAIN PORT DATA TRANSMIT

*IMPLIES NEGATIVE (LOGICAL 0) TRUE INPUT OR OUTPUT

TABLE OF I/O PORTS*

DEVICE* NO.	MANUFACTURER	PORT ADDRESS	FUNCTION
KR3600	STANDARD MICROSYSTEMS CORP.	50H	KEYBOARD CHARACTER (R/O)
BR1941	WESTERN DIGITAL	60H	BAUD RATE GENERATOR (W/O)
8251A	INTEL	40H	AUXILIARY PORT DATA
		41H	AUXILIARY PORT STATUS
		58H	MAIN PORT DATA
		59H	MAIN PORT STATUS
8255	INTEL	68H	8255 PORT A (W/O)
		69H	8255 PORT B (R/O)
		6AH	8255 PORT C (W/O)
		6BH	8255 CONTROL PORT (W/O)
MM58174	NATIONAL SEMICONDUCTOR	31H	DAY/DATE CLOCK TENTHS DIGIT (R/O)
		32H	DAY/DATE CLOCK UNITS OF SECONDS (R/O)
		33H	DAY/DATE CLOCK TENS OF SECONDS (R/O)
		34H	DAY/DATE CLOCK UNITS OF MINUTES (R/W)
		35H	DAY/DATE CLOCK TENS OF MINUTES (R/W)
		36H	DAY/DATE CLOCK UNITS OF HOURS (R/W)
		37H	DAY/DATE CLOCK TENS OF HOURS (R/W)
		38H	DAY/DATE CLOCK UNITS OF DAYS (R/W)
		39H	DAY/DATE CLOCK TENS OF DAYS (R/W)
		3AH	DAY/DATE CLOCK DAY OF THE WEEK (R/W)
		3BH	DAY/DATE CLOCK UNITS OF MONTHS (R/W)
		3CH	DAY/DATE CLOCK TENS OF MONTHS (R/W)
		3DH	DAY/DATE CLOCK LEAP YEAR SETTING (W/O)
		3EH	DAY/DATE CLOCK START/STOP PORT (W/O)

*FOR DETAILED DEVICE INFORMATION, CONSULT MANUFACTURER'S DATA SHEETS.

AUTOLOAD FEATURE

Perhaps you wish for your VPU to perform the same function upon each operating system restart. This is possible with CP/M version 2.2. The command buffer is the area in computer memory where the next command to be executed is placed. In normal CP/M systems this buffer is empty and, upon operating system restart, the system awaits your command. You may alter this if desired, so that the system will execute any program on the disk upon cold or warm reboot.

In order to implement this autoload feature, you have to change the operating system that is stored on the inner two tracks of your diskette. First, make a copy of the program on your distribution diskette that will generate the operating system. In the CompuStar Model 30 (stand-alone mode) this program is called CS30CPM.COM. Using the PIP program, enter the following:

A > **PIP AUTOLOAD.COM = CS30CPM.COM** <cr>

CS30CPM.COM is similar to using the SYSGEN utility, except that no SOURCE DRIVE is specified when using it. After you have made the copy, you will have to alter its command buffer for the autoload capability. The DDT system program will have to be used to do this. It is strongly recommended that you become familiar with the DDT program before attempting to alter the operating system. See the CP/M DYNAMIC DEBUGGING TOOL (DDT) USER'S GUIDE in this manual, for assistance.

Next enter the program 'AUTOLOAD.COM' with the use of DDT. The correct command is:

A > **DDT AUTOLOAD.COM** <cr>

DDT will then load into the computer's memory and read in your AUTOLOAD program. After you have decided on the command you want to be executed upon restart, determine its length. This is done by counting the number of characters in the command. If a file name and/or parameters are included in the command, be sure to include their length(s) in the count. Include any separating spaces. For example, if you wanted the directory display, the command is **DIR**, and its length is 3. If instead you wanted to see a directory display of disk A, the command is **DIR A:** and its length is 6.

The CP/M command buffer begins at location 987H. Use the 'S' command to alter the desired memory locations with your new command. Place the hexadecimal value of the command length in this location. The command itself begins at location 988H, and you may use up to eighty (80) characters from that point for the buffer. Notice that if you go beyond that, you will overwrite the copyright notice in the operating system. At the end of your command, place the null terminator 00H. When inserting the command itself into the memory locations, please note that you must enter hexadecimal numbers for the ASCII values of the letters in the command.

When finished, use the DDT command **D** to display the results of your action. Make any necessary corrections, and then exit to the operating system with CRTL-C. Before you do anything else, you must save the memory contents of the AUTOLOAD program. Using CP/M's 'SAVE' function, enter the following line at the keyboard:

A > **SAVE 48 AUTOLOAD.COM** <cr>

Let's review what we have done so far. First, we made a copy of the operating system, and called it AUTOLOAD.COM. (Incidentally, any other name could have been used as long as the file type is COM). Next, we placed a CP/M command into the CP/M command buffer, starting with the command length in hexadecimal. We ended with a null byte terminator. Then we exited to the operating system and saved the revised program in memory on the disk. Now it is time to generate the new operating system.

Please be sure that the command in the command buffer is what you want your computer to do upon each operating restart, because that is exactly what it will do. Type in the following command at the keyboard:

A > **AUTOLOAD**⟨**cr**⟩

From here the operation will be similar to that of the SYSGEN command. First you will be asked to enter a SOURCE DRIVE. Press the **RETURN** key here; the program itself is carrying the operating system. Next enter the DESTINATION DRIVE. Enter your choice, and press the **RETURN** key when the correct diskette has been inserted in the destination drive. If you are using a new diskette, make certain that it has been formatted with the FORMAT command. When the message FUNCTION COMPLETE is displayed upon the screen, your transfer is done, and you should press the **RETURN** key to reboot the operating system. If you specified drive A as the destination drive, this reboot will incorporate your new modification. If not, replace the diskette in drive A with your destination diskette, and press both RED keys simultaneously. You should now have an operating system with an autoload feature. If not, you probably incorrectly entered the command in the command buffer. Repeat the above procedure if this is the case.

WARNING: If you choose drive A as the destination drive and you made an error in altering the command buffer, this diskette will contain an unusable copy of the operating system. You will have to replace its operating system with a valid copy probably using the SYSGEN command. Therefore, it is recommended that you select drive B as your destination drive when altering the command buffer unless the procedure is being performed on a Model 20 that has no drive B.

Here is a sample session describing the steps needed to alter the command buffer of your operating system. Please carefully read the previous section before attempting to alter this command buffer. Note that all items in bold type are to be typed in by you. Otherwise, the displays are generated by the computer. When you encounter ⟨**cr**⟩, press the **RETURN** key.

A > **PIP AUTOLOAD.COM = CS30CPM.COM[V]** ⟨**cr**⟩

A > **DDT AUTOLOAD.COM**⟨**cr**⟩
DDT VER 1.4
NEXT PC
3100 0100

– S987⟨**cr**⟩
0987 00 **06**⟨**cr**⟩
0988 20 **44**⟨**cr**⟩
0989 20 **49**⟨**cr**⟩
098A 20 **52**⟨**cr**⟩
098B 20 **20**⟨**cr**⟩
098C 20 **41**⟨**cr**⟩
098D 20 **3A**⟨**cr**⟩
098E 20 **00**⟨**cr**⟩
098F 20 **.**⟨**cr**⟩
-CONTROL-C
A > SAVE 48 AUTOLOAD.COM ⟨**cr**⟩

A > **AUTOLOAD** ⟨**cr**⟩

SYSGEN VER 1.X
SOURCE DRIVE NAME (OR RETURN TO SKIP) ⟨**cr**⟩
DESTINATION DRIVE NAME (OR RETURN TO REBOOT) **B**⟨**cr**⟩
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT) ⟨**cr**⟩

A >

(Now replace the diskette in drive B into drive A, and depress RED keys.)

This loads and executes the modified operating system. Use the diskette now in drive A as your normal operating system diskette. The function you specified will be run each time a cold or warm boot is performed.

KEY CLICK

The key click feature is designed to provide a tone with each key depression. The purpose of the feedback is to allow faster data entry by informing the operator whenever a key is depressed. This feature can be easily selected during terminal operation or can be automatically selected upon system power-up.

To enable the feedback feature, simply type a Control-B (02H). This will 'toggle' the key click feature and turn it on if it is off, or vice versa. The CONFIGUR program will permit you to set the click on or off on system power-up, and hence, relieve you of any further action.

KEY REPEAT (Stand-Alone Mode Only)

When a key remains depressed for more than one second, the key value will repeat at a rate of approximately 30 per second. This will allow faster data entry for applications such as word processing, text editing, and program displays where a 'banner' is required.

TYPE-AHEAD (Stand-Alone Mode Only)

The input on DOS version 1.X is saved if the operator enters data faster than the computer can accept it. Up to 128 characters are stored when typed, and delivered only when needed. It is now possible to enter commands to an application program as it is being loaded from the disk and not lose any characters. Your input will appear after the program has been loaded, and the program will execute the commands as if you had just entered them. If you type more than 128 characters ahead of the computer system, the bell will ring. This indicates that the buffer is full, and further typing will be ignored by the system.

NOTE: It should be noted that some programs will not work with the type-ahead feature. An example is the DIR command, which displays the directory contents of a diskette. By definition, a directory display is interrupted if a key is depressed during the display. If the DIR command receives a key from the type-ahead feature, it doesn't know if the key was just entered, or if it came from the buffer. In either case, the display is disrupted and a character is lost. Experiment with the system to see which programs will not tolerate type-ahead.

In the event that an error is made, the type-ahead buffer can be erased by depressing the **CONTROL** key and the **1** key (on the alphanumeric keyboard only, not the numeric keypad) simultaneously.

CONTROLLING THE VIDEO DISPLAY

The CompuStar VPU allows the user a great degree of flexibility in controlling the video display. The user can control where the display is on the screen and the appearance of the displayed information.

Data positioning can be effected either by absolute cursor addressing or memory-mapping. Display appearance is controlled by two factors. First, the CompuStar VPU has an optional character set available to the user. Alternating character sets as well as video attributes can be effected on a character by character basis. Second, there are four video attributes. These are:

* Blinking.
* Half-intensity.
* Underlining.
* Reverse Video.

Memory-mapping means that a portion of the memory is devoted to use by the screen display. The RAM memory location F800H marks the beginning of screen area and this area extends through location FFFFH. This memory area is not available for program or data storage.

The CRT controller performs a direct memory access (DMA) cycle to obtain the screen data, relieving the CPU of most screen related functions. When the CRT controller receives certain inputs, the display is affected.

There are two main types of inputs that are meaningful to the CRT controller: escape sequences and control codes. An escape sequence is noted when the ASCII representation of ESC (1BH) is received by the CRT controller and followed by other characters.

A control code is noted when the CTRL key of the keyboard is held down while another key is depressed. The CTRL key functions somewhat like the SHIFT key does.

Escape Sequences

The following is a list of escape sequences that have meaning to the CRT controller.
NOTE: " ~ " is equivalent to ASCII code 7E (Hex) or 126 (decimal).

SEQUENCE	MEANING
ESC Y row column	Absolute cursor addressing. The cursor is positioned to the row and column as shown in the screen layout chart in this section.
ESC ~ K	Erase to end of line. Data is erased from the current cursor position through the end of the current line.
ESC ~ k	Erase to end of screen. Data is erased from the current cursor position through the end of the current screen.
ESC ~ E	Display control characters. The transparent mode of operation is enabled which means that control codes not normally shown on the screen will be displayed.
ESC ~ D	Disable display of control characters.
ESC ~ B	Turns the blinking video attribute on.
ESC ~ b	Turns the blinking video attribute off.
ESC ~ H	Turns the half-intensity attribute on.
ESC ~ h	Turns the half-intensity attribute off.
ESC ~ U	Turns the underlining attribute on.
ESC ~ u	Turns the underlining attribute off.
ESC ~ R	Turns the reverse video attribute on.
ESC ~ r	Turns the reverse video attribute off.
ESC ~ A	Makes the entire screen non-reverse video.

Escape Sequences (continued)

SEQUENCE	MEANING
ESC ~ a	Makes the entire screen reverse video.
ESC ~ N	Normalizes. Turns all attribute indicators (B, H, U, R) off if they are on, beginning with the next character.
ESC ~ g	Displays the entirety of what is on the screen as its corresponding alternate(s) from the secondary character set. This only works if the secondary character EPROM is installed.
ESC ~ G	This reverses the effects of the ESC ~ g escape sequence preceeding.
ESC ~ S	This sequence reverses the primary and secondary assignments of the character sets when an alternate (secondary) character set is installed. If set A is primary and set B is secondary, this sequence will cause B to be primary and A to be secondary.
ESC ~ s	This reverses the effect on the ESC ~ S sequence preceeding.

NOTE: Of the escape sequences discussed, the S, G, a, and A options affect the entire screen including data on the screen entered prior to this sequence.

NOTE: The high order bit of the ASCII character is what controls switching between primary and secondary character sets. A "0" is the high order bit selects primary set. A "1" in the high order bit selects the secondary set.

NOTE: The CP/M Console Command Processor upshifts all lower case characters typed. Therefore any escape sequences using lower case characters cannot be entered from the Console Command Mode.

Control Codes

The following is a list of the control codes that have meaning to the CRT controller.

CODE	MEANING
CTRL-A	Home cursor — The cursor is positioned at row 1, column 1.
CTRL-F	Cursor forward — The cursor is moved one space to the right.
CTRL-G	Ring bell — The audio indicator is activated.
CTRL-H	Cursor back — The cursor is moved one space to the left.
CTRL-K	Cursor up — The cursor is moved up one line.
CTRL-J	Cursor down — The cursor is positioned down one line.
CTRL-I	Tabbing — The cursor is positioned to the next tab (modulo-8) position.
CTRL-L	Clear screen — Erases the data on the screen and the cursor is moved to row 1, column 1, its home position.

Control Codes (continued)

CODE	MEANING
CTRL-R	Redisplays current CP/M command line.
CTRL-X	Clears current CP/M command line.
CTRL-@	Page off/on — video display scrolling is enabled or disabled. Valid during operator input only and not subject to user program control.
CTRL-1	Clears type ahead buffer.

Video Attributes

Attributes are set by the CompuStar VPU when particular escape sequences (see previous list) are received by the Console Out routine of the CP/M BIOS (and subsequently the CRT controller). The escape sequence consists of an ESC(ape), followed by a TILDE, followed by a hexadecimal representation of the attribute desired.

The hexadecimal format is 1B 7E NN where NN assumes the following value as desired.

42H - B	61H - a
62H - b	41H - A
48H - H	4EH - N
68H - h	67H - g
55H - U	47H - G
75H - u	53H - S
52H - R	73H - s
72H - r	

The following program written in MBASIC language distributed with your CompuStar VPU, shows a technique for attribute manipulation.

```
100 CY = 20
110 CX = 5
120 REM Clear screen and then show some of the CompuStar VPU video attributes.
130 REM
140 PRINT CHR$(12)
150 REM first line is normal
160 GOSUB 510
170 PRINT "CompuStar Video Attribute Demo"
180 REM
190 REM Now turn on inverse video and reprint line.
200 REM
210 CX = 7:GOSUB 510
220 PRINT CHR$(27); " ~ R";
230 PRINT "CompuStar Video Attribute Demo"
240 REM
250 REM Now turn on half intensity and reprint line
260 REM
270 REM
280 CX = 9:GOSUB 510
290 PRINT CHR$(27);" ~ H";
300 PRINT "CompuStar Video Attribute Demo"
```

```
310  REM
320  REM Turn inverse back off and turn underlining on
330  REM
340  CX = 11:GOSUB 510
350  PRINT CHR$(27); " ~ r"; CHR$(27);" ~ U";
360  PRINT "CompuStar Video Attribute Demo"
380  REM
390  REM Turn half intensity off but leave underlining on
420  REM
430  CX = 13:GOSUB 510
450  PRINT CHR$(27);" ~ h";
470  PRINT "CompuStar Video Attribute Demo"
471  REM
472  REM Now normalize the video attributes
473  REM
474  PRINT CHR$(27):" ~ N"
480  PRINT
490  PRINT
500  END
510  Print CHR$(11)
520  PRINT CHR(27);"Y"; CHR$(CX + 31);CHR$ = (CY + 31);
530  RETURN
```

Cursor Positioning for Display Control

Cursor positioning is easily accomplished using the ESC Y row column escape sequence. The proper row, column coordinates can be determined by referencing the CompuStar VPU screen layout in this section.

The example program that follows, written in MBASIC, shows one method of accomplishing screen control.

Memory Map/Screen Initialization

This BASIC program fragment will clear the screen and set the "HOME" position to be memory address &HF800. The user can then "POKE" characters into the next 1,920 locations of screen memory.

NOTE: Line number 1110 leaves the cursor at the top of the screen. The cursor can, at this time, be moved where the user wishes with standard escape sequence cursor positioning commands.

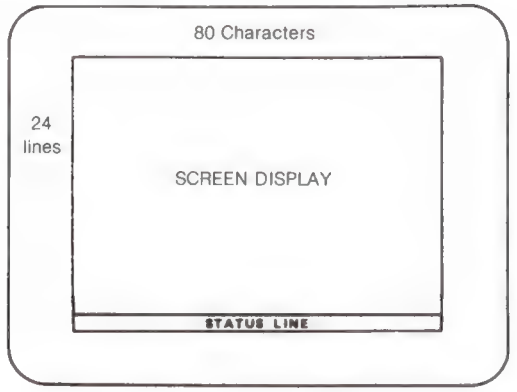
```
1090   PRINT CHR$(12);
1100   FOR I = 1 to 23:PRINT " " :Next I
1110   PRINT " "; CHR$(1);
1120   RETURN
```

The next example program, also written in MBASIC, shows an example of cursor positioning.

```
100    REM Clear the Screen
110    PRINT CHR$(12)
120    REM Position the cursor at row 20 column 30
130    CX = 20
140    CY = 30
150    GOSUB 2000
160    PRINT "* --POSITION 20, 30"
```

```
170   REM Position the cursor at row 5 column 20
180   CX = 5
190   CY = 20
200   GOSUB 2000
210   PRINT ''* –POSITION 5, 20''
220   REM Home cursor and then end
230   PRINT CHR$(1);
240   END
2000  REM Cursor Positioning Subroutine
2010  REM
2020  REM This subroutine clears the MBASIC line output character counter
2030  REM and then positions the cursor at the locations specified by the
2040  REM variables CX and CY where CX is the line number and CY is the
2050  REM column number. These variables must be set by the program before
2060  REM entering the subroutine
2070  REM
2080  REM NOTE: Home position on the screen is row 1 column 1
2090  REM
2100  PRINT CHR$(11)
2110  PRINT CHR$(27);''Y'';CHR$(CX + 31);CHR$(CY + 31);
2120  RETURN
```

SUPERBRAIN SCREEN LAYOUT



This Screen Format of the SuperBrain display area provides an easy method of locating and addressing specific screen positions.

Using the ESC, Y, r, c command, locate both the row character (r = 1 - 24) and the column (c = 1 - 80) characters. Example:

ROW	COLUMN	COMMAND
1 (Home)	1	ESC Y sp sp
2	5	ESC y ! $
20	50	ESC Y 3 Q

An application programmer may find it helpful to maintain a table of row and column numbers with their respective addressing characters as shown on this Screen Format. This will provide quick and easy access to specific screen positions.

Interpreting the ASCII Code Chart

The figure below illustrates a conventionally arranged ASCII code chart divided into three sections corresponding to control codes (column 0 to 1) upper case characters (columns 2, 3, 4, and 5), and lower case characters (columns 4 and 5).

b_7 b_6 b_5 →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
Bits ↓	b_4 ↓	b_3 ↓	b_2 ↓	b_1 ↓	column → / row ↓	0	1	2	3	4	5	6	7
	0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
	0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
	0	0	1	0	2	STX	DC2	"	2	B	R	b	r
	0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
	0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
	0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
	0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
	0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
	1	0	0	0	8	BS	CAN	(	8	H	X	h	x
	1	0	0	1	9	HT	EM	)	9	I	Y	i	y
	1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
	1	0	1	1	11	VT	ESC	+	;	K	[	k	{
	1	1	0	0	12	FF	FS	,	<	L	\	l	¦
	1	1	0	1	13	CR	GS	—	=	M	]	m	}
	1	1	1	0	14	SO	RS	.	>	N	^	n	~
	1	1	1	1	15	SI	US	/	?	O	—	o	DEL

If the terminal is set for upper case operation only (CAPS LOCK), lower case alpha characters from the keyboard are automatically translated and displayed as their upper equivalents (columns 4 and 5). If the DEL code is received, it is ignored. Lower case characters received from the input RS-232C port are displayed as lower case.

The seven bit binary code for each character is divided into two parts in this chart. A four-bit number represents the four least significant bits (B1, B2, B3, B4) and a three-bit number represents the three most significant bits (B5, B6, B7). The chart above also is divided into 8 columns and 16 rows. This offers two ways of indicating a particular character's code. The character code is indicated as either a seven-bit binary number or as a column/row number in decimal notation. For example, the character M is represented by the binary number 1001101 or the alternative 4/13 notation. Similarly, the control code VT is represented by the code 0001011 or the alternative 0/11 notation.

For the CompuStar VPU, the high order bit is used to determine switching between the primary and secondary character sets. This eighth (or high order) bit is not shown in this chart but exists and can be manipulated from user programs.

CONTROL CODE CHART

The following is a list of the hexadecimal equivalents of the control codes. The CONFIGUR program accepts only hexadecimal values when reassigning the keypad, so these are listed as a programmer convenience. Use caution when reassigning the values on the keypad, and recall that you may enter 'R' to restore the pad to its original configuration if you desire.

Ctrl-A	01H	Ctrl-J	0AH	Ctrl-S	13H
Ctrl-B	02H	Ctrl-K	0BH	Ctrl-T	14H
Ctrl-C	03H	Ctrl-L	0CH	Ctrl-U	15H
Ctrl-D	04H	Ctrl-M	0DH	Ctrl-V	16H
Ctrl-E	05H	Ctrl-N	0EH	Ctrl-W	17H
Ctrl-F	06H	Ctrl-O	0FH	Ctrl-X	18H
Ctrl-G	07H	Ctrl-P	10H	Ctrl-Y	19H
Ctrl-H	08H	Ctrl-Q	11H	Ctrl-Z	1AH
Ctrl-I	09H	Ctrl-R	12H		

After all corrections have been entered, pressing the **RETURN** key will save your new parameters on the disk. This must be done at the main menu of selections. Then press both RED keys when instructed to force a 'cold boot' of the operating system and properly load your new changed parameters. Make certain that the silver "write protect" tab is removed, if present, from the diskette in drive A so that the updated parameters can be saved.

Control codes are not displayable unless in the transparent mode. Some of these codes affect the state of the terminal when they are received by the display electronics. For example, the code SOH causes the cursor to go to the home position, and code DC2 turns on the printer port. Codes which have no defined function in the CompuStar VPU software are ignored if received. The set of 64 upper case alphanumeric characters is sometimes referred to as "compressed ASCII".

WORDSTAR CONSIDERATIONS FOR COMPUSTAR VPU

This is to set up a version of WordStar for the CompuStar VPU that uses the CompuStar VPU in a memory mapped mode. The following variable names are in Appendix D of the WordStar manual ("Terminal Patch Area"). The following items need to be set as indicated:

```
0264    UCRPOS    JMP     0304H     ; User cursor positioning
                                    ;      routine for memory map
                                    ;      operation.

02A4    INISUB    JMP     02E0H     ; Jump to the CompuStar VPU
                                    ;      initialization routine.

02B0    MEMAPV    DB      0FFH      ; Turn memory map mode on
02B1    MEMADR    DB      0F800H    ; Address of video screen RAM.

02E0    MORPAT    CALL    02E8H     ; Initialize the CompuStar VPU
02E3              DEC     A         ;      video memory map.
02E4              CALL    0239H
02E7              RET
02E8              XRA     A
02E9              LXI     H,0E434H
02EC              MVI     B,18H
02EE              MOV     M,A
02EF              INX     H
02F0              DEC     B
02F1              JNZ     02EEH
02F4              LXI     H,0000
```

```
02F7            SHLD    0E400H
02FA            SHLD    0E414H
02FD            SHLD    0E416H
0300            SHLD    0E412H
0303            RET
0304            XCHG
0305            JMP     0300H
```

If the user has the reverse video character set EPROM installed in the secondary character set EPROM position, WordStar can also highlight certain items by setting the following value:

```
02B3  HIBIV     DB      0FFH        ; Highlight using the high bit
```

APPENDIX A

COMPUSTAR NETWORKING OPERATIONS

APPENDIX A-1

DISK STORAGE SYSTEM

96 MB

NOTE: THIS APPENDIX WILL BE FURNISHED WITH THE PURCHASE OF THE 96 MB CONVERSION KIT

APPENDIX A-2

DSS-10
DISK STORAGE SYSTEM

NOTE: THIS APPENDIX WILL BE FURNISHED WITH THE PURCHASE OF THE DSS-10

APPENDIX A-3

DISK STORAGE SYSTEM

144 MB

NOTE: THIS APPENDIX WILL BE FURNISHED WITH THE PURCHASE OF THE 144 MB DISK DRIVE

APPENDIX A-4

DISK STORAGE SYSTEM

144 MB TAPE BACK-UP UNIT

NOTE: THIS APPENDIX WILL BE FURNISHED WITH THE PURCHASE OF THE 144 MB TAPE BACK-UP UNIT

APPENDIX B

CP/M

CP/M®

OPERATING SYSTEM

MANUAL

DIGITAL RESEARCH™
P.O. Box 579
Pacific Grove, California 93950

COPYRIGHT

Copyright © 1976, 1977, 1978, 1979, and 1982 by Digital Research. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of Digital Research, Post Office Box 579, Pacific Grove, California 93950.

DISCLAIMER

Digital Research makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Digital Research reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Digital Research to notify any person of such revision or changes.

TRADEMARKS

CP/M is a registered trademark of Digital Research. MP/M, MAC, and SID are trademarks of Digital Research. Z-80 is a trademark of Zilog, Inc.

First Printing: July 1982

CONTENTS

1 CP/M FEATURES AND FACILITIES 1

1.1 Introduction 1
1.2 Functional Description 3
1.2.1 General Command Structure 3
1.2.2 File References 3
1.3 Switching Disks 5
1.4 Built-in Commands 6
1.4.1 ERA 6
1.4.2 DIR 6
1.4.3 REN 7
1.4.4 SAVE 8
1.4.5 TYPE 8
1.4.6 USER 8
1.5 Line Editing and Output Control 9
1.6 Transient Commands 9
1.6.1 STAT 10
1.6.1 ASM 15
1.6.3 LOAD 16
1.6.4 PIP 17
1.6.5 ED 23
1.6.6 SYSGEN 24
1.6.7 SUBMIT 25
1.6.8 DUMP 27
1.6.9 MOVCPM 27
1.7 BDOS Error Messages 29
1.8 Operation of CP/M on the MDS 30

2 ED 33

2.1 Introduction to ED 33
2.1.1 ED Operation 33
2.1.2 Text Transfer Functions 35
2.1.3 Memory Buffer Organization 35
2.1.4 Line Numbers and ED Start Up 36
2.1.5 Memory Buffer Operation 37
2.1.6 Command Strings 38
2.1.7 Text Search and Alteration 39
2.1.8 Source Libraries 42
2.1.9 Repetitive Command Execution 42
2.2 ED Error Conditions 43
2.3 Control Characters and Commands 44

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

3 CP/M ASSEMBLER 47

3.1 Introduction 47
3.2 Program Format 48
3.3 Forming the Operand 49
3.3.1 Labels 50
3.3.2 Numeric Constants 50
3.3.3 Reserved Words 50
3.3.4 String Constants 51
3.3.5 Arithmetic and Logical Operators 52
3.3.6 Precedence of Operators 52
3.4 Assembler Directives 53
3.4.1 The ORG Directive 54
3.4.2 The END Directive 54
3.4.3 The EQU Directive 55
3.4.4 The SET Directive 55
3.4.5 The IF and ENDIF Directives 56
3.4.6 The DB Directive 57
3.4.7 The DW Directive 57
3.4.8 The DS Directive 57
3.5 Operation Codes 58
3.5.1 Jumps, Calls, and Returns 58
3.5.2 Immediate Operand Instructions 59
3.5.3 Increment and Decrement Instructions 60
3.5.4 Data Movement Instructions 60
3.5.5 Arithmetic Logic Unit Operations 61
3.5.6 Control Instructions 62
3.6 Error Messages 62
3.7 A Sample Session 63

4 CP/M DYNAMIC DEBUGGING TOOL 69

4.1 Introduction 69
4.2 DDT Commands 71
4.2.1 The A (Assembly) Command 71
4.2.2 The D (Display) Command 72
4.2.3 The F (Fill) Command 72
4.2.4 The G (Go) Command 72
4.2.5 The I (Input) Command 73
4.2.6 The L (List) Command 74
4.2.7 The M (Move) Command 74
4.2.8 The R (Read) Command 74
4.2.9 The S (Set) Command 75
4.2.10 The T (Trace) Command 75
4.2.11 The U (Untrace) Command 76
4.2.12 The X (Examine) Command 76
4.3 Implementation Notes 77
4.4 An Example 78

5 CP/M 2 SYSTEM INTERFACE 89

5.1 Introduction 89
5.2 Operating System Call Conventions 91
5.3 A Sample File-to-File Copy Program 110
5.4 A Sample File Dump Utility 113
5.5 A Sample Random Access Program 117
5.6 System Function Summary 124

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

6 CP/M ALTERATION 127

6.1 Introduction 127
6.2 First Level System Regeneration 128
6.3 Second Level System Generation 131
6.4 Sample GETSYS and PUTSYS Program 134
6.5 Diskette Organization 136
6.6 The BIOS Entry Points 137
6.7 A Sample BIOS 143
6.8 A Sample Cold Start Loader 143
6.9 Reserved Locations in Page Zero 144
6.10 Disk Parameter Tables 145
6.11 The DISKDEF Macro Library 148
6.12 Sector Blocking and Deblocking 152

APPENDIXES

A The MDS Basic I/O System (BIOS) 153
B A Skeletal CBIOS 175
C A Skeletal GETSYS/PUTSYS Program 187
D The MDS-800 Cold Start Loader for CP/M 2 191
E A Skeletal Cold Start Loader 197
F CP/M Disk Definition Library 201
G Blocking and Deblocking Algorithms 209
H Glossary 219
I CP/M Messages 235

INDEX 245

FIGURES

2.1 Overall ED Operation 34
2.2 Memory Buffer Organization 34
2.3 Logical Organization of Memory Buffer 36

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

1

CP/M Features and Facilities

1.1 Introduction

CP/M is a monitor control program for microcomputer system development that uses floppy disks or Winchester hard disks for backup storage. Using a computer system based upon Intel's 8080 microcomputer, CP/M provides a general environment for program construction, storage, and editing, along with assembly and program check-out facilities. An important feature of CP/M is that it can be easily altered to execute with any computer configuration that uses an Intel 8080 (or Zilog Z-80) Central Processing Unit and has at least 20K bytes of main memory with up to 16 diskette drives. A detailed discussion of the modifications required for any particular hardware environment is given in Chapter 6. Although the standard Digital Research version operates on a single-density Intel MDS 800, several different hardware manufacturers support their own input-output drivers for CP/M.

The CP/M monitor provides rapid access to programs through a comprehensive file management package. The file subsystem supports a named file structure, allowing dynamic allocation of file space as well as sequential and random file access. Using this file system, a large number of programs can be stored in both source and machine-executable form.

CP/M 2 is a high-performance, single-console operating system that uses table-driven techniques to allow field reconfiguration to match a wide variety of disk capacities. All fundamental file restrictions are removed, maintaining upward compatibility from previous versions of release 1. Features of CP/M 2 include field specification of one to sixteen logical drives, each containing up to eight megabytes. Any particular file can reach the full drive size with the capability of expanding to thirty-two megabytes in future releases. The directory size can be field-configured to contain any reasonable number of entries, and each file is optionally tagged with read/only and system attributes. Users of CP/M 2 are physically separated by user numbers, with facilities for file copy operations from one user area to another. Powerful relative-record random access functions are present in CP/M 2 that provide direct access to any of the 65536 records of an eight-megabyte file.

CP/M also supports a powerful context editor, Intel-compatible assembler, and debugger subsystems. Optional software includes a powerful Intel-compatible macro assembler, symbolic debugger, along with various high-level languages. When coupled

with CP/M's Console Command Processor, the resulting facilities equal or excel similar large computer facilities.

CP/M is logically divided into several distinct parts:

BIOS	Basic I/O System (hardware-dependent)
BDOS	Basic Disk Operating System
CCP	Console Command Processor
TPA	Transient Program Area

The BIOS provides the primitive operations necessary to access the diskette drives and to interface standard peripherals (teletype, CRT, paper tape reader/punch, and user-defined peripherals). They can be tailored by the user for any particular hardware environment by "patching" this portion of CP/M. The BDOS provides disk management by controlling one or more disk drives containing independent file directories. The BDOS implements disk allocation strategies that provide fully dynamic file construction while minimizing head movement across the disk during access. The BDOS has entry points that include the following primitive operations, which can be programmatically accessed:

SEARCH	Look for a particular disk file by name.
OPEN	Open a file for further operations.
CLOSE	Close a file after processing.
RENAME	Change the name of a particular file.
READ	Read a record from a particular file.
WRITE	Write a record to a particular file.
SELECT	Select a particular disk drive for further operations.

The CCP provides a symbolic interface between the user's console and the remainder of the CP/M system. The CCP reads the console device and processes commands, which include listing the file directory, printing the contents of files, and controlling the operation of transient programs, such as assemblers, editors, and debuggers. The standard commands that are available in the CCP are listed in Section 1.2.1.

The last segment of CP/M is the area called the Transient Program Area (TPA). The TPA holds programs that are loaded from the disk under command of the CCP. During program editing, for example, the TPA holds the CP/M text editor machine code and data areas. Similarly, programs created under CP/M can be checked out by loading and executing these programs in the TPA.

Any or all of the CP/M component subsystems can be "overlaid" by an executing program. That is, once a user's program is loaded into the TPA, the CCP, BDOS, and BIOS areas can be used as the program's data area. A "bootstrap" loader is programmatically accessible whenever the BIOS portion is not overlaid; thus, the user program need only branch to the bootstrap loader at the end of execution and the complete CP/M monitor is reloaded from disk.

The CP/M operating system is partitioned into distinct modules, including the BIOS portion that defines the hardware environment in which CP/M is executing. Thus, the standard system is easily modified to any nonstandard environment by changing the peripheral drivers to handle the custom system.

1.2 Functional Description

The user interacts with CP/M primarily through the CCP, which reads and interprets commands entered through the console. In general, the CCP addresses one of several disks that are on-line (the standard system addresses up to sixteen different disk drives). These disk drives are labeled A through P. A disk is "logged in" if the CCP is currently addressing the disk. To clearly indicate which disk is the currently logged disk, the CCP always prompts the operator with the disk name followed by the symbol ">" indicating that the CCP is ready for another command. Upon initial start-up, the CP/M system is brought in from disk A, and the CCP displays the message

```
CP/M VER m.m
```

where m.m is the CP/M version number. All CP/M systems are initially set to operate in a 20K memory space, but can be easily reconfigured to fit any memory size on the host system (see Section 1.6.9). Following system sign-on, CP/M automatically logs in disk A, prompts the user with the symbol "A>" (indicating that CP/M is currently addressing disk "A"), and waits for a command. The commands are implemented at two levels: built-in commands and transient commands.

1.2.1 General Command Structure

Built-in commands are a part of the CCP program itself, while transient commands are loaded into the TPA from disk and executed. The built-in commands are

ERA	Erase specified files.
DIR	List file names in the directory.
REN	Rename the specified file.
SAVE	Save memory contents in a file.
TYPE	Type the contents of a file on the logged disk.

Most of the commands reference a particular file or group of files. The form of a file reference is specified below.

1.2.2 File References

A file reference identifies a particular file or group of files on a particular disk attached to CP/M. These file references are either "unambiguous" (ufn) or "ambiguous" (afn). An unambiguous file reference uniquely identifies a single file, while an ambiguous file reference is satisfied by a number of different files.

File references consist of two parts: the primary filename and the filetype. Although the filetype is optional, it usually is generic; that is, the filetype "ASM," for example, is used to denote that the file is an assembly language source file, while the primary filename distinguishes each particular source file. The two names are separated by a ".", as shown below:

```
filename.typ
```

where filename is the primary filename of eight characters or less, and typ is the filetype of no more than three characters. As mentioned above, the name

```
filename
```

is also allowed and is equivalent to a filetype consisting of three blanks. The characters used in specifying an unambiguous file reference cannot contain any of the special characters

```
< > . , ; : = ? * [ ] _ % | ( ) / \
```

while all alphanumerics and remaining special characters are allowed.

An ambiguous file reference is used for directory search and pattern matching. The form of an ambiguous file reference is similar to an unambiguous reference, except the symbol "?" can be interspersed throughout the primary and secondary names. In various commands throughout CP/M, the "?" symbol matches any character of a file name in the "?" position. Thus, the ambiguous reference

```
X?Z.C?M
```

is satisfied by the unambiguous file names

```
XYZ.COM
```

and

```
X3Z.CAM
```

Note that the ambiguous reference

```
*.*
```

is equivalent to the ambiguous file reference

```
????????.???
```

while

```
filename.*
```

and

```
*.typ
```

are abbreviations for

```
filename.???
```

and

```
????????.typ
```

respectively. As an example,

```
A>DIR *.*
```


ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

is interpreted by the CCP as a command to list the names of all disk files in the directory, while

```
A>DIR X.Y
```

searches only for a file by the name X.Y. Similarly, the command

```
A>DIR X?Y.C?M
```

causes a search for all (unambiguous) file names on the disk that satisfy this ambiguous reference.

The following file names are valid unambiguous file references:

X	XYZ	GAMMA
X.Y	XYZ.COM	GAMMA.1

As an added convenience, the programmer can generally specify the disk drive name along with the file name. In this case, the drive name is given as a letter A through P followed by a colon (:). The specified drive is then "logged in" before the file operation occurs. Thus, the following are valid file names with disk name prefixes:

A:X.Y	B:XYZ	C:GAMMA
P:XYZ.COM	B:X. A?M	C:*. ASM

All alphabetic lower case letters in file and drive names are translated to upper case when they are processed by the CCP.

1.3 Switching Disks

The operator can switch the currently logged disk by typing the disk drive name (A through P) followed by a colon (:) when the CCP is waiting for console input. Thus, the sequence of prompts and commands below can occur after the CP/M system is loaded from disk A:

```
CP/M VER 2.2
A>DIR                    List all files on disk A.
A: SAMPLE ASM SAMPLE PRN
A>B:                     Switch to disk B.
B>DIR *. ASM             List all "ASM" files on B.
B: DUMP ASM FILES ASM
B>A:                     Switch back to A.
```

1.4 Built-in Commands

The file and device reference forms described can now be used to fully specify the structure of the built-in commands. The user should assume the following abbreviations in the description below:

ufn	unambiguous file reference
afn	ambiguous file reference

Recall that the CCP always translates lower case characters to upper case characters internally. Thus, lower case alphabetics are treated as if they are upper case in command names and file references.

1.4.1 ERA afn

The ERA (erase) command removes files from the currently logged in disk (i.e., the disk name currently prompted by CP/M preceding the ">"). The files that are erased are those that satisfy the ambiguous file reference afn. The following examples illustrate the use of ERA:

ERA X.Y	The file named X.Y on the currently logged disk is removed from the disk directory and the space is returned.
ERA X.*	All files with primary name X are removed from the current disk.
ERA *. ASM	All files with secondary name ASM are removed from the current disk.
ERA X?Y.C?M	All files on the current disk that satisfy the ambiguous reference X?Y.C?M are deleted.
ERA *.*	Erase all files on the current disk (in this case the CCP prompts the console with the message ALL FILES (Y/N)? that requires a Y response before files are actually removed).
ERA B:*.PRN	All files on drive B that satisfy the ambiguous reference ????????.PRN are deleted, independently of the currently logged disk.

1.4.2 DIR afn

The DIR (directory) command causes the names of all files that satisfy the ambiguous file name afn to be listed at the console device. As a special case, the command

```
DIR
```


ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

lists the files on the currently logged disk (the command "DIR" is equivalent to the command "DIR *.*"). Valid DIR commands are

```
DIR X.Y
DIR X?Z.C?M
DIR ??.Y
```

Similar to other CCP commands, the afn can be preceded by a drive name. The following DIR commands cause the selected drive to be addressed before the directory search takes place.

```
DIR B:
DIR B:X.Y
DIR B:*. A?M
```

If no files on the selected diskette satisfy the directory request, the message "NO FILE" is typed at the console.

1.4.3 REN ufn1=ufn2

The REN (rename) command allows the user to change the names of files on disk. The file satisfying ufn2 is changed to ufn1. The currently logged disk is assumed to contain the file to rename (ufn2). The user can also type a left-directed arrow instead of the equal sign if the console supports this graphic character. Examples of the REN command are

REN X.Y=Q.R	The file Q.R is changed to X.Y.
REN XYZ.COM=XYZ.XXX	The file XYZ.XXX is changed to XYZ.COM.

The operator precedes either ufn1 or ufn2 (or both) by an optional drive address. If ufn1 is preceded by a drive name, then ufn2 is assumed to exist on the same drive. Similarly, if ufn2 is preceded by a drive name, then ufn1 is assumed to exist on that drive as well. The same drive must be specified in both cases if both ufn1 and ufn2 are preceded by drive names. The REN commands below illustrate this format.

REN A:X.ASM=Y.ASM	The file Y.ASM is changed to X.ASM on drive A.
REN B:ZAP.BAS=ZOT.BAS	The file ZOT.BAS is changed to ZAP.BAS on drive B.
REN B:A.ASM=B:A.BAK	The file A.BAK is renamed to A.ASM on drive B.

If ufn1 is already present, the REN command will respond with the error "FILE EXISTS" and not perform the change. If ufn2 does not exist on the specified diskette, the message "NO FILE" is printed at the console.

1.4.4 SAVE n ufn

The SAVE command places n pages (256-byte blocks) onto disk from the TPA and names this file ufn. In the CP/M distribution system, the TPA starts at 100H (hexadecimal) which is the second page of memory. The SAVE command must specify 2 pages of memory if the user's program occupies the area from 100H through 2FFH. The machine code file can be subsequently loaded and executed. Examples are

SAVE 3 X.COM	Copies 100H through 3FFH to X.COM.
SAVE 40 Q	Copies 100H through 28FFH to Q (note that 28 is the page count in 28FFH, and that 28H = 2*16+8 = 40 decimal).
SAVE 4 X.Y	Copies 100H through 4FFH to X.Y.

The SAVE command can also specify a disk drive in the ufn portion of the command, as shown below.

SAVE 10 B:ZOT.COM	Copies 10 pages (100H through 0AFFH) to the file ZOT.COM on drive B.

1.4.5 TYPE ufn

The TYPE command displays the contents of the ASCII source file ufn on the currently logged disk at the console device. Valid TYPE commands are

```
TYPE X.Y
TYPE X.PLM
TYPE XXX
```

The TYPE command expands tabs (clt-I characters), assuming tab positions are set at every eighth column. The ufn can also reference a drive name.

TYPE B:X.PRN	The file X.PRN from drive B is displayed.

1.4.6 USER n

The USER command allows maintenance of separate files in the same directory and takes the form

```
USER n
```

where n is an integer value in the range 0 to 15. On cold start, the operator is automatically "logged" into user area number 0, which is compatible with standard CP/M 1 directories. The operator may issue the USER command at any time to move to another logical area within the same directory. Drives that are logged-in while addressing one user number are automatically active when the operator moves to another; a user number is simply a prefix that accesses particular directory entries on the active disks.

The active user number is maintained until changed by a subsequent USER command, or until a cold start when user 0 is again assumed.

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

1.5 Line Editing and Output Control

The CCP allows certain line editing functions while typing command lines.

ctl-C	CP/M system reboot when typed at start of line.
ctl-E	Physical end of line: carriage is returned, but line is not sent until the carriage return key is depressed.
ctl-H	Backspace one character position.
ctl-J	Terminate current input (line feed).
ctl-M	Terminate current input (carriage return).
ctl-R	Retype current command line: types a "clean line" following character deletion with rubouts.
ctl-U	Delete the entire line typed at the console.
ctl-X	Same as ctl-U.
ctl-Z	End input from the console (used in PIP and ED).
rub/del	Delete and echo the last character typed at the console.

The control functions ctl-P and ctl-S affect console output.

ctl-P	Copy all subsequent console output to the currently assigned list device (see Section 1.6.1). Output is sent to the list device and the console device until the next ctl-P is typed.
ctl-S	Stop the console output temporarily. Program execution and output continue when the next character is typed at the console (e.g., another ctl-S). This feature stops output on high speed consoles, such as CRT's, in order to view a segment of output before continuing.

The ctl-key sequences are obtained by depressing the control and letter keys simultaneously. Further, CCP command lines are generally up to 255 characters in length; they are not acted upon until the carriage return key is typed.

1.6 Transient Commands

Transient commands are loaded from the currently logged disk and executed in the TPA. The transient commands for execution under the CCP are below. Additional functions are easily defined by the user (see Section 1.6.3).

STAT	List the number of bytes of storage remaining on the currently logged disk, provide statistical information about particular files, and display or alter device assignment.
ASM	Load the CP/M assembler and assemble the specified program from disk.
LOAD	Load the file in Intel "HEX" machine code format and produce a file in machine executable form that can be loaded into the TPA (this loaded program becomes a new command under the CCP).
DDT	Load the CP/M debugger into TPA and start execution.

PIP	Load the Peripheral Interchange Program for subsequent disk file and peripheral transfer operations.
ED	Load and execute the CP/M text editor program.
SYSGEN	Create a new CP/M system diskette.
SUBMIT	Submit a file of commands for batch processing.
DUMP	Dump the contents of a file in hex.
MOVCPM	Regenerate the CP/M system for a particular memory size.

Transient commands are specified in the same manner as built-in commands, and additional commands are easily defined by the user. For convenience, the transient command can be preceded by a drive name that causes the transient to be loaded from the specified drive into the TPA for execution. Thus, the command

```
B:STAT
```

causes CP/M to temporarily "log in" drive B for the source of the STAT transient, and then return to the original logged disk for subsequent processing.

The basic transient commands are listed in detail below.

1.6.1 STAT

The STAT command provides general statistical information about file storage and device assignment. It is initiated by typing one of the following forms:

```
STAT
STAT "command line"
```

Special forms of the "command line" allow the current device assignment to be examined and altered. The various command lines that can be specified are shown, with an explanation of each form to the right.

STAT

If the user types an empty command line, the STAT transient calculates the storage remaining on all active drives, and prints a message

```
d: R/W, SPACE: nnnK
```

or

```
d: R/O, SPACE: nnnK
```

for each active drive d:, where R/W indicates the drive can be read or written, and R/O indicates the drive is read only (a drive becomes R/O by explicitly setting it to read only, as shown below, or by inadvertently changing diskettes without performing a warm start). The space remaining on the diskette in drive d: is given in kilobytes by nnn.

STAT d:

If a drive name is given, then the drive is selected before the storage is computed. Thus, the command "STAT B:" could be issued while logged into drive A, resulting in the message

```
BYTES REMAINING ON B: nnnK
```

STAT afn

The command line can also specify a set of files to be scanned by STAT. The files that satisfy afn are listed in alphabetical order, with storage requirements for each file under the heading

```
RECS BYTS EX D:FILENAME.TYP
rrrr  bbbK  ee  d:filename.typ
```

where rrrr is the number of 128-byte records allocated to the file, bbb is the number of kilobytes allocated to the file (bbb=rrrr*128/1024), ee is the number of 16K extensions (ee=bbb/16), d is the drive name containing the file (A...P), filename is the (up to) eight-character primary filename, and typ is the (up to) three-character filetype. After listing the individual files, the storage usage is summarized.

STAT d:afn

The drive name can be given ahead of the afn. The specified drive is first selected, and the form "STAT afn" is executed.

STAT d:=R/O

This form sets the drive given by d to read only, remaining in effect until the next warm or cold start takes place. When a disk is read only, the message

```
BDOS ERR ON d: READ ONLY
```

will appear if there is an attempt to write to the read-only disk d:. CP/M waits until a key is depressed before performing an automatic warm start (at which time the disk becomes R/W).

The STAT command allows control over the physical to logical device assignment (see the IOBYTE function described in Chapters 5 and 6). There are four logical peripheral devices that are, at any particular instant, each assigned one of several physical peripheral devices. The four logical devices are

CON:	The system console device (used by CCP for communication with the operator)
RDR:	The paper tape reader device
PUN:	The paper tape punch device
LST:	The output list device

The actual devices attached to any particular computer system are driven by subroutines in the BIOS portion of CP/M. Thus, the logical RDR: device, for example, could

actually be a high speed reader, teletype reader, or cassette tape. To allow some flexibility in device naming and assignment, several physical devices are defined below:

TTY:	Teletype device (slow speed console)
CRT:	Cathode ray tube device (high speed console)
BAT:	Batch processing (console is current RDR:, output goes to current LST: device)
UC1:	User-defined console
PTR:	Paper tape reader (high speed reader)
UR1:	User-defined reader #1
UR2:	User-defined reader #2
PTP:	Paper tape punch (high speed punch)
UP1:	User-defined punch #1
UP2:	User-defined punch #2
LPT:	Line printer
UL1:	User-defined list device #1

It is emphasized that the physical device names may or may not actually correspond to devices that the names imply. That is, the PTP: device may be implemented as a cassette write operation if the user wishes. The exact correspondence and driving subroutine is defined in the BIOS portion of CP/M. In the standard distribution version of CP/M, these devices correspond to their names on the MDS 800 development system.

The command

```
STAT VAL:
```

produces a summary of the available status commands, resulting in the output

```
Temp R/O Disk d:$R/O
Set Indicator: filename.typ $R/O $R/W $SYS $DIR
Disk Status: DSK: d:DSK
Iobyte Assign:
```

which gives an instant summary of the possible STAT commands and shows the permissible logical-to-physical device assignments:

```
CON: = TTY: CRT: BAT: UC1:
RDR: = TTY: PTR: UR1: UR2:
PUN: = TTY: PTP: UP1: UP2:
LST: = TTY: CRT: LPT: UL1:
```

The logical device to the left takes any of the four physical assignments shown to the right. The current logical to physical mapping is displayed by typing the command

```
STAT DEV:
```


ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

producing a list of each logical device to the left and the current corresponding physical device to the right. For example, the list might appear as

```
CON: = CRT:
RDR: = UR1:
PUN: = PTP:
LST: = TTY:
```

The current logical to physical device assignment is changed by typing a STAT command of the form

```
STAT ld1 = pd1,  ld2 = pd2 , ... , ldn = pdn
```

where ld1 through ldn are logical device names and pd1 through pdn are compatible physical device names (i.e., ldi and pdi appear on the same line in the "VAL:" command shown above). Valid STAT commands that change the current logical to physical device assignments are

```
STAT CON:=CRT:
STAT PUN: = TTY:, LST:=LPT:, RDR:=TTY:
```

The command form

```
STAT d:filename.typ $S
```

where "d:" is an optional drive name and "filename.typ" is an unambiguous or ambiguous file name, produces the output display format

Size	Recs	Bytes	Ext Acc
48	48	6k	1 R/O A:ED.COM
55	55	12k	1 R/O (A:PIP.COM)
65536	128	16k	2 R/W A:X.DAT

where the $S parameter causes the "Size" field to be displayed. (Without the $S, the Size field is skipped, but the remaining fields are displayed.) The Size field lists the virtual file size in records, while the "Recs" field sums the number of virtual records in each extent. For files constructed sequentially, the Size and Recs fields are identical. The "Bytes" field lists the actual number of bytes allocated to the corresponding file. The minimum allocation unit is determined at configuration time; thus, the number of bytes corresponds to the record count plus the remaining unused space in the last allocated block for sequential files. Random access files are given data areas only when written, so the Bytes field contains the only accurate allocation figure. In the case of random access, the Size field gives the logical end-of-file record position and the Recs field counts the logical records of each extent. (Each of these extents, however, may contain unallocated "holes" even though they are added into the record count.) The "Ext" field counts the number of physical extents allocated to the file. The Ext count corresponds to the number of directory entries given to the file. Depending on allocation size, there can be up to 128K bytes (8 logical extents) directly addressed by a single directory entry. (In a special case, there are actually 256K bytes that can be directly addressed by a physical extent.)

The Acc field gives the R/O or R/W file indicator that is changed using the commands shown. Similarly, the parentheses shown about the PIP.COM filename indicate that it

has the "system" indicator set, so that it will not be listed in DIR commands. The four command forms

```
STAT d:filename.typ $R/O
STAT d:filename.typ $R/W
STAT d:filename.typ $SYS
STAT d:filename.typ $DIR
```

set or reset various permanent file indicators. The R/O indicator places the file (or set of files) in a read-only status until changed by a subsequent STAT command. The R/O status is recorded in the directory with the file so that it remains R/O through intervening cold start operations. The R/W indicator places the file in a permanent read/write status. The SYS indicator attaches the system indicator to the file, while the DIR command removes the system indicator. The "filename.typ" may be ambiguous or unambiguous, but files whose attributes are changed are listed at the console when the change occurs. The drive name denoted by "d:" is optional.

When a file is marked R/O, subsequent attempts to erase or write into the file result in a terminal BDOS message

```
BDOS Err on d: File R/O
```

The BDOS waits for a console input before performing a subsequent warm start (a "return" is sufficient). The command form

```
STAT d:DSK:
```

lists the drive characteristics of the disk named by "d:" that is in the range A:, B:, ..., P:. The drive characteristics are listed in the format

```
    d: Drive Characteristics
65536: 128 Byte Record Capacity
 8192: Kilobyte Drive Capacity
  128: 32 Byte Directory Entries
    0: Checked Directory Entries
 1024: Records/ Extent
  128: Records/ Block
   58: Sectors/ Track
    2: Reserved Tracks
```

where "d:" is the selected drive, followed by the total record capacity (65536 is an eight-megabyte drive), followed by the total capacity listed in kilobytes. The directory size is listed next, followed by the "checked" entries. The number of checked entries is usually identical to the directory size for removable media, because this mechanism is used to detect changed media during CP/M operation without an intervening warm start. For fixed media, the number is usually zero, because the media are not changed without at least a cold or warm start. The number of records per extent determines the addressing capacity of each directory entry (1024 times 128 bytes, or 128K in the previous example). The number of records per block shows the basic allocation size (in the example, 128 records/block times 128 bytes per record, or 16K bytes per block). The listing is then followed by the number of physical sectors per track and the number of reserved tracks.

For logical drives that share the same physical disk, the number of reserved tracks can be quite large because this mechanism is used to skip lower-numbered disk areas allocated to other logical disks. The command form

```
STAT DSK:
```

produces a drive characteristics table for all currently active drives. The final STAT command form is

```
STAT USR:
```

which produces a list of the user numbers that have files on the currently addressed disk. The display format is

```
Active User: 0
Active Files: 0 1 3
```

where the first line lists the currently addressed user number, as set by the last CCP USER command, followed by a list of user numbers scanned from the current directory. In this case, the active user number is 0 (default at cold start), with three user numbers that have active files on the current disk. The operator can subsequently examine the directories of the other user numbers by logging-in with USER 1 or USER 3 commands, followed by a DIR command at the CCP level.

1.6.2 ASM ufn

The ASM command loads and executes the CP/M 8080 assembler. The ufn specifies a source file containing assembly language statements where the filetype is assumed to be ASM and is not specified. The following ASM commands are valid:

```
ASM X
ASM GAMMA
```

The two-pass assembler is automatically executed. Assembly errors that occur during the second pass are printed at the console.

The assembler produces a file

```
X.PRN
```

where X is the primary name specified in the ASM command. The PRN file contains a listing of the source program (with imbedded tab characters if present in the source program), along with the machine code generated for each statement and diagnostic error messages, if any. The PRN file is listed at the console using the TYPE command, or sent to a peripheral device using PIP (see Section 1.6.4). The user should note that the PRN file contains the original source program, augmented by miscellaneous assembly information in the leftmost 16 columns (program addresses and hexadecimal machine code, for example). The PRN file serves as a backup for the original source file. If the source file is accidentally removed or destroyed, the PRN file can be edited (see Chapter 2) by removing the leftmost 16 characters of each line. This is done by issuing a single editor "macro" command. The resulting file is identical to the original source file and can be renamed (REN) from PRN to ASM for subsequent editing and assembly. The file

```
X.HEX
```

is also produced, which contains 8080 machine language in Intel "HEX" format suitable for subsequent loading and execution (see Section 1.6.3). For complete details of CP/M's assembly language program, see Chapter 3.

The source file for assembly is taken from an alternate disk by prefixing the assembly language file name by a disk drive name. The command

```
ASM B:ALPHA
```

loads the assembler from the currently logged drive and processes the source program ALPHA.ASM on drive B. The HEX and PRN files are also placed on drive B in this case.

1.6.3 LOAD ufn

The LOAD command reads the file ufn, which is assumed to contain "HEX" format machine code, and produces a memory image file that can subsequently be executed. The file name ufn is assumed to be of the form

```
X.HEX
```

and only the filename X need be specified in the command. The LOAD command creates a file named

```
X.COM
```

that marks it as containing machine executable code. The file is actually loaded into memory and executed when the user types the filename X immediately after the prompting character ">" printed by the CCP.

Generally the CCP reads the filename X following the prompting character and looks for a built-in function name. If no function name is found, the CCP searches the system disk directory for a file by the name

```
X.COM
```

If found, the machine code is loaded into the TPA, and the program executes. Thus, the user need only LOAD a hex file once; it can be subsequently executed any number of times by typing the primary name. In this way the user can "invent" new commands in the CCP. (Initialized disks contain the transient commands as COM files, which are deleted at the user's option.) The operation takes place on an alternate drive if the file name is prefixed by a drive name. Thus

```
LOAD B:BETA
```

brings the LOAD program into the TPA from the currently logged disk and operates upon drive B after execution begins.

The user should note that the BETA.HEX file must contain valid Intel format hexadecimal machine code records (as produced by the ASM program, for example) that begin at 100H of the TPA. The addresses in the hex records must be in ascending order; gaps in unfilled memory regions are filled with zeroes by the LOAD command as the hex records are read. Thus, LOAD must be used only for creating CP/M standard "COM" files that operate in the TPA. Programs that occupy regions of memory other than the TPA are loaded under DDT.

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

1.6.4 PIP

PIP is the CP/M Peripheral Interchange Program that implements the basic media conversion operations necessary to load, print, punch, copy, and combine disk files. The PIP program is initiated by typing one of the following forms:

(1) PIP

(2) PIP 'command line'

In both cases PIP is loaded into the TPA and executed. In form (1), PIP reads command lines directly from the console, prompted with the "*" character, until an empty command line is typed (i.e., a single carriage return is issued by the operator). Each successive command line causes some media conversion to take place according to the rules shown below. Form (2) of the PIP command is equivalent to the first, except that the single command line given with the PIP command is automatically executed, and PIP terminates immediately with no further prompting of the console for input command lines. The form of each command line is

destination = source#1, source#2, ... , source#n

where "destination" is the file or peripheral device to receive the data and "source#1, ..., source#n" is a series of one or more files or devices that are copied from left to right to the destination.

When multiple files are given in the command line (i.e., $n > 1$), the individual files are assumed to contain ASCII characters, with an assumed CP/M end-of-file character (ctl-Z) at the end of each file (see the O parameter to override this assumption). Lower case ASCII alphabetics are internally translated to upper case to be consistent with CP/M file and device name conventions. Finally, the total command line length cannot exceed 255 characters (ctl-E can be used to force a physical carriage return for lines that exceed the console width).

The destination and source elements are unambiguous references to CP/M source files with or without a preceding disk drive name. That is, any file can be referenced with a preceding drive name (A: through P:) that defines the particular drive where the file may be obtained or stored. When the drive name is not included, the currently logged disk is assumed. The destination file can also appear as one or more of the source files, in which case the source file is not altered until the entire concatenation is complete. If it already exists, the destination file is removed if the command line is properly formed (it is not removed if an error condition arises). The following command lines (with explanations to the right) are valid as input to PIP:

X=Y	Copy to file X from file Y, where X and Y are unambiguous file names; Y remains unchanged.
X=Y,Z	Concatenate files Y and Z and copy to file X, with Y and Z unchanged.
X.ASM=Y.ASM,Z.ASM,FIN.ASM	Create the file X.ASM from the concatenation of the Y, Z, and FIN files with type ASM.
NEW.ZOT=B:OLD.ZAP	Move a copy of OLD.ZAP from drive B to the currently logged disk; name the file NEW.ZOT.

B:A.U = B:B.V,A:C.W,D.X	Concatenate file B.V from drive B with C.W from drive A and D.X. from the logged disk; create the file A.U on drive B.

For convenience, PIP allows abbreviated commands for transferring files between disk drives. The abbreviated forms are

PIP d:=afn

PIP d_1:=d_2:afn

PIP ufn = d_2:

PIP d_1:ufn = d_2:

The first form copies all files from the currently logged disk that satisfy the afn to the same files on drive d (d = A . . . P). The second form is equivalent to the first, where the source for the copy is drive d_2 (d_2 = A . . . P). The third form is equivalent to the command "PIP d_1:ufn=d_2:ufn" that copies the file given by ufn from drive d_2 to the file ufn on drive d_1:. The fourth form is equivalent to the third, where the source disk is explicitly given by d_2:.

The source and destination disks must be different in all of these cases. If an afn is specified, PIP lists each ufn that satisfies the afn as it is being copied. If a file exists by the same name as the destination file, it is removed on successful completion of the copy and replaced by the copied file.

The following PIP commands give examples of valid disk-to-disk copy operations:

B:=*.COM	Copy all files that have the secondary name "COM" to drive B from the current drive.
A:=B:ZAP.*	Copy all files that have the primary name "ZAP" to drive A from drive B.
ZAP.ASM=B:	Equivalent to ZAP.ASM=B:ZAP.ASM
B:ZOT.COM=A:	Equivalent to B:ZOT.COM=A:ZOT.COM
B:=GAMMA.BAS	Same as B:GAMMA.BAS=GAMMA.BAS
B:=A:GAMMA.BAS	Same as B:GAMMA.BAS=A:GAMMA.BAS

PIP allows reference to physical and logical devices that are attached to the CP/M system. The device names are the same as given under the STAT command, along with a number of specially named devices. The logical devices given in the STAT command are

CON: (console), RDR: (reader), PUN: (punch), and LST: (list)

while the physical devices are

TTY: (console, reader, punch, or list)

CRT: (console, or list), UC1: (console)

PTR: (reader), UR1: (reader), UR2: (reader)

PTP: (punch), UP1: (punch), UP2: (punch)

LPT: (list), UL1: (list)

(The "BAT:" physical device is not included, since this assignment is used only to indicate that the RDR: and LST: devices are used for console input/output.)

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

The RDR, LST, PUN, and CON devices are all defined within the BIOS portion of CP/M, and are easily altered for any particular I/O system. (The current physical device mapping is defined by IOBYTE; see Chapter 6 for a discussion of this function). The destination device must be capable of receiving data (i.e., data cannot be sent to the punch), and the source devices must be capable of generating data (i.e., the LST: device cannot be read).

The additional device names that can be used in PIP commands are

NUL:	Send 40 "nulls" (ASCII 0's) to the device (this can be issued at the end of punched output).
EOF:	Send a CP/M end-of-file (ASCII ctl-Z) to the destination device (sent automatically at the end of all ASCII data transfers through PIP).
INP:	Special PIP input source that can be patched into the PIP program: PIP gets the input data character-by-character by CALLing location 103H, with data returned in location 109H (parity bit must be zero).
OUT:	Special PIP output destination that can be patched into the PIP program: PIP CALLs location 106H with data in register C for each character to transmit. The user should note that locations 109H through 1FFH of the PIP memory image are not used and can be replaced by special purpose drivers using DDT (see Chapter 4).
PRN:	Same as LST: except that tabs are expanded at every eighth character position, lines are numbered, and page ejects are inserted every 60 lines with an initial eject (same as using PIP options [t8np]).

File and device names can be interspersed in the PIP commands. In each case, the specific device is read until end-of-file (ctl-Z for ASCII files, and end-of-data for non-ASCII disk files). Data from each device or file are concatenated from left to right until the last data source has been read. The destination device or file is written using the data from the source files, and an end-of-file character (ctl-Z) is appended to the result for ASCII files. If the destination is a disk file, a temporary file is created ($$$ secondary name) that is changed to the actual file name only on successful completion of the copy. Files with the extension "COM" are always assumed to be non-ASCII.

The copy operation can be aborted at any time by depressing any key on the keyboard (a return suffices). PIP will respond with the message "ABORTED" to indicate that the operation has not been completed. If any operation is aborted, or if an error occurs during processing, PIP removes any pending commands that were set up while using the SUBMIT command.

PIP performs a special function if the destination is a disk file with type "HEX" (an Intel hex-formatted machine code file), and the source is an external peripheral device, such as a paper tape reader. In this case, the PIP program checks to ensure that the source file contains a properly formed hex file, with legal hexadecimal values and checksum records. When an invalid input record is found, PIP reports an error message at the console and waits for corrective action. It is usually sufficient to open the reader and rerun a section of the tape (pull the tape back about 20 inches). When the tape is ready for the reread, a single carriage return is typed at the console, and PIP will attempt another read. If the tape position cannot be properly read, the user continues the read (by typing a return following the error message), and enters the record manually with the ED program after the disk file is constructed. For convenience, PIP allows the end-of-file to be entered from the console if the source file is an RDR: device. In this case, the PIP program reads the device and monitors the keyboard. If ctl-Z is typed at the keyboard the read operation is terminated normally.

Valid PIP commands are

PIP LST: = X.PRN	Copy X.PRN to the LST device and terminate the PIP program.
PIP	Start PIP for a sequence of commands (PIP prompts with "*").
*CON:=X.ASM,Y.ASM,Z.ASM	Concatenate three ASM files and copy to the CON device.
*X.HEX=CON:,Y.HEX,PTR:	Create a HEX file by reading the CON (until a ctl-Z is typed), followed by data from Y.HEX and PTR until a ctl-Z is encountered.
(carriage return)	Single carriage return stops PIP.
PIP PUN:=NUL:,X.ASM,EOF:,NUL:	Send 40 nulls to the punch device; copy the X.ASM file to the punch, followed by an end-of-file (ctl-Z) and 40 more null characters.

The user can also specify one or more PIP parameters, enclosed in left and right square brackets, separated by zero or more blanks. Each parameter affects the copy operation, and the enclosed list of parameters must immediately follow the affected file or device. Generally, each parameter can be followed by an optional decimal integer value (the S and Q parameters are exceptions). Valid PIP parameters are

B	Block mode transfer: data are buffered by PIP until an ASCII x-off character (ctl-S) is received from the source device. This allows transfer of data to a disk file from a continuous reading device, such as a cassette reader. Upon receipt of the x-off, PIP clears the disk buffers and returns for more input data. The amount of data that can be buffered depends on the memory size of the host system (PIP will issue an error message if the buffers overflow).
Dn	Delete characters that extend past column n in the transfer of data to the destination from the character source. This parameter is generally used to truncate long lines that are sent to a (narrow) printer or console device.
E	Echo all transfer operations to the console as they are being performed.
F	Filter form feeds from the file. All imbedded form feeds are removed. The P parameter can be used simultaneously to insert new form feeds.
Gn	Get File from user number n (n in the range 0-15).
H	HEX data transfer: all data are checked for proper Intel hex file format. Nonessential characters between hex records are removed during the copy operation. The console will be prompted for corrective action in case errors occur.
I	Ignore ":00" records in the transfer of Intel hex format file (the I parameter automatically sets the H parameter).
L	Translate upper case alphabetics to lower case.

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

N	Add line numbers to each line transferred to the destination, starting at one and incrementing by 1. Leading zeroes are suppressed, and the number is followed by a colon. If N2 is specified, leading zeroes are included and a tab is inserted following the number. The tab is expanded if T is set.
O	Object file (non-ASCII) transfer: the normal CP/M end-of-file is ignored.
Pn	Include page ejects at every n lines (with an initial page eject). If n = 1 or is excluded altogether, page ejects occur every 60 lines. If the F parameter is used, form feed suppression takes place before the new page ejects are inserted.
Qstz	Quit copying from the source device or file when the string s (terminated by ctl-Z) is encountered.
R	Read system files.
Sstz	Start copying from the source device when the string s (terminated by ctl-Z) is encountered. The S and Q parameters can be used to "abstract" a particular section of a file (such as a subroutine). The start and quit strings are always included in the copy operation. If the user selects form (2) of the PIP command, the CCP translates strings following the S and Q parameters to upper case. Form (1) of the PIP invocation does not perform the automatic upper case translation. (1) PIP (2) PIP 'command line'
Tn	Expand tabs (ctl-I characters) to every nth column during the transfer of characters to the destination from the source.
U	Translate lower case alphabetics to upper case during the copy operation.
V	Verify that data have been copied correctly by rereading after the write operation (the destination must be a disk file).
W	Write over R/O files without console interrogation.
Z	Zero the parity bit on input for each ASCII character.

Valid PIP commands that specify parameters in the file transfer are

PIP X.ASM=B:[v]	Copy X.ASM from drive B to the current drive and verify that the data were properly copied.
PIP LPT:=X.ASM[nt8u]	Copy X.ASM to the LPT: device; number each line, expand tabs to every eighth column, and translate lower case alphabetics to upper case.
PIP PUN:=X.HEX[i],Y.ZOT[h]	First copy X.HEX to the PUN: device and ignore the trailing ":00" record in X.HEX; continue the transfer of data by reading Y.ZOT, which contains HEX records, including any ":00" records it contains.

PIP X.LIB = Y.ASM [sSUBRI:^Z qJMP L3^Z]	Copy from the file Y.ASM into the file X.LIB. Start the copy when the string "SUBRI:" has been found, and quit copying after the string "JMP L3" is encountered.
PIP PRN:=X.ASM[p50]	Send X.ASM to the LST: device with line numbers, tabs expanded to every eighth column, and page ejects at every 50th line. The assumed parameter list for a PRN file is nt8p60; p50 overrides the default value.

Under normal operation, PIP will not overwrite a file that is set to a permanent R/O status. If an attempt is made to overwrite an R/O file, the prompt

```
DESTINATION FILE IS R/O, DELETE (Y/N)?
```

is issued. If the operator responds with the character "y" the file is overwritten. Otherwise, the response

```
** NOT DELETED **
```

is issued, the file transfer is skipped, and PIP continues with the next operation in sequence. To avoid the prompt and response in the case of R/O file overwrite, the command line can include the W parameter

```
PIP A:=B:*.COM[W]
```

which copies all nonsystem files to the A drive from the B drive and overwrites any R/O files in the process. If the operation involves several concatenated files, the W parameter need only be included with the last file in the list, as in the example

```
PIP A.DAT = B.DAT,F:NEW.DAT,G:OLD.DAT[W]
```

Files with the system attribute can be included in PIP transfers if the R parameter is included; otherwise, system files are not recognized. The command line

```
PIP ED.COM = B:ED.COM[R]
```

for example, reads the ED.COM file from the B drive, even if it has been marked as an R/O and system file. The system file attributes are copied, if present.

Downward compatibility with previous versions of CP/M is only maintained if the file does not exceed one megabyte, no file attributes are set, and the file is created by user 0. If compatibility is required with nonstandard (e.g., "double density") versions of 1.4, it may be necessary to select 1.4 compatibility mode when constructing the internal disk parameter block. (See Chapter 6 and refer to Section 6.10, which describes BIOS differences.)

Note: To copy files into another user area, PIP.COM must be located in that user area. Follow the procedure shown below to make a copy of PIP.COM in another user area.

USER 0	Log-in user 0.
DDT PIP.COM (note PIP size s)	Load PIP to memory.
G0	Return to CCP.
USER 3	Log-in user 3.
SAVE s PIP.COM	

where s is the integral number of memory "pages" (256-byte segments) occupied by PIP. The number s can be determined when PIP.COM is loaded under DDT, by referring to the value under the NEXT display. If, for example, the next available address is 1D00, then PIP.COM requires 1C hexadecimal pages (or 1 times 16 + 12 = 28 pages), and the value of s is 28 in the subsequent save. Once PIP is copied in this manner, it can be copied to another disk belonging to the same user number through normal PIP transfers.

1.6.5 ED ufn

The ED program is the CP/M system context editor that allows creation and alteration of ASCII files in the CP/M environment. Complete details of operation are given in Chapter 2. ED allows the operator to create and operate upon source files that are organized as a sequence of ASCII characters, separated by end-of-line characters (a carriage-return line-feed sequence). There is no practical restriction on line length (no single line can exceed the size of the working memory) that is defined by the number of characters typed between carriage returns. The ED program has a number of commands for character string searching, replacement, and insertion that are useful in creation and correction of programs or text files under CP/M. Although the CP/M has a limited memory work space area (approximately 5000 characters in a 20K CP/M system), the file size that can be edited is not limited, since data are easily "paged" through this work area.

If it does not exist, ED creates the specified source file and opens the file for access. If the source file does exist (see the A command), the programmer "appends" data for editing. The appended data can then be displayed, altered, and written from the work area back to the disk (see the W command). Particular points in the program can be automatically paged and located by context (see the N command), allowing easy access to particular portions of a large file.

Given that the operator has typed

```
ED X.ASM
```

the ED program creates an intermediate work file with the name

```
X.$$$
```

to hold the edited data during the ED run. Upon completion of ED, the X.ASM file (original file) is renamed to X.BAK, and the edited work file is renamed to X.ASM. Thus, the X.BAK file contains the original (unedited) file, and the X.ASM file contains the newly edited file. The operator can always return to the previous version of a file by removing the most recent version and renaming the previous version. If the current X.ASM file has been improperly edited, the sequence of commands below will reclaim the backup file.

```
DIR X.*          Check to see that BAK file is available.
ERA X.ASM        Erase most recent version.
REN X.ASM=X.BAK  Rename the BAK file to ASM.
```

The operator can abort the edit at any point (reboot, power failure, ctl-C, or Q command) without destroying the original file. In this case, the BAK file is not created and the original file is always intact.

The ED program allows the user to edit the source on one disk and create the backup file on another disk. This form of the ED command is

```
ED ufn d:
```

where ufn is the name of the file to edit on the currently logged disk and d is the name of an alternate drive. The ED program reads and processes the source file and writes the new file to drive d using the name ufn. After processing, the original file becomes the backup file. If the operator is addressing disk A, the following command is valid:

```
ED X.ASM B:
```

This edits the file X.ASM on drive A, creating the new file X.$$$ on drive B. After a successful edit, A:X.ASM is renamed to A:X.BAK, and B:X.$$$ is renamed to B:X.ASM. For convenience the currently logged disk becomes drive B at the end of the edit. The user should note that if a file named B:X.ASM exists before the editing begins, the message

```
FILE EXISTS
```

is printed at the console as a precaution against accidentally destroying a source file. The operator first erases the existing file and then restarts the edit operation.

Similar to other transient commands, editing can take place on a drive different from the currently logged disk by preceding the source file name by a drive name. Examples of valid edit requests are

ED A:X.ASM	Edit the file X.ASM on drive A, with new file and backup on drive A.
ED B:X.ASM A:	Edit the file X.ASM on drive B to the temporary file X.$$$ on drive A. After editing, change X.ASM on drive B to X.BAK and change X.$$$ on drive A to X.ASM.

1.6.6 SYSGEN

The SYSGEN transient command allows generation of an initialized diskette containing the CP/M operating system. The SYSGEN program prompts the console for commands by interacting as shown.

SYSGEN cr	Initiate the SYSGEN program.
SYSGEN VERSION m.m	SYSGEN sign-on message.
SOURCE DRIVE NAME (OR RETURN TO SKIP)	Respond with the drive name (one of the letters A, B, C, or D) of the disk containing a CP/M system, usually A. If a copy of CP/M already exists in memory due to a MOVCPM command, type a carriage return only. Typing a drive name d will cause the response:
SOURCE ON d THEN TYPE RETURN	Place a diskette containing the CP/M operating system on drive d (d is one of A, B, C, or D). Answer by typing a carriage return when ready.

FUNCTION COMPLETE	System is copied to memory. SYSGEN will then prompt with:
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)	If a diskette is being initialized, place the new disk into a drive and answer with the drive name. Otherwise, type a cr and the system will reboot from drive A. Typing drive name d will cause SYSGEN to prompt with:
DESTINATION ON d THEN TYPE RETURN	Place new diskette into drive d; type return when ready.
FUNCTION COMPLETE	New diskette is initialized in drive d.

The "DESTINATION" prompt will be repeated until a single carriage return is typed at the console, so that more than one disk can be initialized.

Upon completion of a successful system generation, the new diskette contains the operating system and only the built-in commands are available. A factory-fresh, IBM-compatible diskette appears to CP/M as a diskette with an empty directory; therefore, the operator must copy the appropriate COM files from an existing CP/M diskette to the newly constructed diskette using the PIP transient.

The user can copy all files from an existing diskette by typing the PIP command

```
PIP B: = A: *.*[v]
```

which copies all files from disk drive A to disk drive B and verifies that each file has been copied correctly. The name of each file is displayed at the console as the copy operation proceeds.

The user should note that a SYSGEN does not destroy the files that already exist on a diskette; it only constructs a new operating system. If a diskette is being used only on drives B through P and will never be the source of a bootstrap operation on drive A, the SYSGEN need not take place.

1.6.7 SUBMIT ufn parm#1 ... parm#n

The SUBMIT command allows CP/M commands to be batched for automatic processing. The ufn given in the SUBMIT command must be the file name of a file that exists on the currently logged disk, with an assumed file type of "SUB." The SUB file contains CP/M prototype commands with possible parameter substitution. The actual parameters parm#1 ... parm#n are substituted into the prototype commands, and, if no errors occur, the file of substituted commands are processed sequentially by CP/M.

The prototype command file is created using the ED program, with interspersed "$" parameters of the form

```
$1 $2 $3 ... $n
```

corresponding to the number of actual parameters that will be included when the file is submitted for execution. When the SUBMIT transient is executed, the actual parameters parm#1 ... parm#n are paired with the formal parameters $1 ... $n in the prototype commands. If the numbers of formal and actual parameters do not correspond, the

submit function is aborted with an error message at the console. The SUBMIT function creates a file of substituted commands with the name

```
$$$.SUB
```

on the logged disk. When the system reboots (at the termination of the SUBMIT), this command file is read by the CCP as a source of input rather than the console. If the SUBMIT function is performed on any disk other than drive A, the commands are not processed until the disk is inserted into drive A and the system reboots. The user can abort command processing at any time by typing a rubout when the command is read and echoed. In this case the $$$.SUB file is removed and the subsequent commands come from the console. Command processing is also aborted if the CCP detects an error in any of the commands. Programs that execute under CP/M can abort processing of command files when error conditions occur by erasing any existing $$$.SUB file.

To introduce dollar signs into a SUBMIT file, the user may type a "$$" which reduces to a single "$" within the command file. An up-arrow symbol "^" may precede an alphabetic character x, which produces a single ctl-x character within the file.

The last command in a SUB file can initiate another SUB file, allowing chained batch commands.

Suppose the file ASMBL.SUB exists on disk and contains the prototype commands

```
ASM $1
DIR $1.*
ERA *.BAK
PIP $2:=$1.PRN
ERA $1.PRN
```

and the command

```
SUBMIT ASMBL X PRN
```

is issued by the operator. The SUBMIT program reads the ASMBL.SUB file, substituting "X" for all occurrences of $1 and "PRN" for all occurrences of $2. This results in a $$$.SUB file containing the commands

```
ASM X
DIR X.*
ERA *.BAK
PIP PRN:=X.PRN
ERA X.PRN
```

which are executed in sequence by the CCP.

The SUBMIT function can access a SUB file on an alternate drive by preceding the file name by a drive name. Submitted files are only acted upon when they appear on drive A. Thus it is possible to create a submitted file on drive B that is executed at a later time when inserted in drive A.

An additional utility program called XSUB extends the power of the SUBMIT facility to include line input to programs as well as the console command processor. The XSUB command is included as the first line of the submit file. When it is executed, XSUB self-relocates directly below the CCP. All subsequent submit command lines are processed by XSUB so that programs that read buffered console input (BDOS function 10)

receive their input directly from the submit file. For example, the file SAVER.SUB can contain the submit lines

```
XSUB
DDT
I$1.COM
R
G0
SAVE 1 $2.COM
```

with a subsequent SUBMIT command

```
A>SUBMIT SAVER PIP Y
```

that substitutes X for $1 and Y for $2 in the command stream. The XSUB program loads, followed by DDT, which is sent to the command lines PIP.COM, R, and G0, thus returning to the CCP. The final command SAVE 1 Y.COM is processed by the CCP.

The XSUB program remains in memory and prints the message

```
(xsub active)
```

on each warm start operation to indicate its presence. Subsequent submit command streams do not require the XSUB, unless an intervening cold start has occurred. The user should note that XSUB must be loaded after the optional CP/M DESPOOL utility, if both are to run simultaneously.

1.6.8 DUMP ufn

The DUMP program types the contents of the disk file (ufn) at the console in hexadecimal form. The file contents are listed sixteen bytes at a time, with the absolute byte address listed to the left of each line in hexadecimal. Long typeouts can be aborted by pushing the rubout key during printout. (The source listing of the DUMP program is given in Chapter 5 as an example of a program written for the CP/M environment.)

1.6.9 MOVCPM

The MOVCPM program allows the user to reconfigure the CP/M system for any particular memory size. Two optional parameters can be used to indicate the desired size of the new system and the disposition of the new system at program termination. If the first parameter is omitted or an "*" is given, the MOVCPM program will reconfigure the system to its maximum size, based upon the kilobytes of contiguous RAM in the host system (starting at 0000H). If the second parameter is omitted, the system is executed, but not permanently recorded; if "*" is given, the system is left in memory, ready for a SYSGEN operation. The MOVCPM program relocates a memory image of CP/M and places this image in memory in preparation for a system generation operation. The command forms are

MOVCPM	Relocate and execute CP/M for management of the current memory configuration (memory is examined for contiguous RAM, starting at 100H). On

	completion of the relocation, the new system is executed but not permanently recorded on the diskette. The system that is constructed contains a BIOS for the Intel MDS 800.
MOVCPM n	Create a relocated CP/M system for management of an n kilobyte system (n must be in the range of 20 to 64), and execute the system as described.
MOVCPM * *	Construct a relocated memory image for the current memory configuration, but leave the memory image in memory in preparation for a SYSGEN operation.
MOVCPM n *	Construct a relocated memory image for an n kilobyte memory system, and leave the memory image in preparation for a SYSGEN operation.

The command

```
MOVCPM * *
```

for example, constructs a new version of the CP/M system and leaves it in memory, ready for a SYSGEN operation. The message

```
READY FOR 'SYSGEN' OR
'SAVE 34 CPMxx.COM'
```

is printed at the console upon completion, where xx is the current memory size in kilobytes. The operator can then type

SYSGEN	Start the system generation.
SOURCE DRIVE NAME (OR RETURN TO SKIP)	Respond with a carriage return to skip the CP/M read operation since the system is already in memory as a result of the previous MOVCPM operation.
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)	Respond with B to write new system to the diskette in drive B. SYSGEN will prompt with:
DESTINATION ON B, THEN TYPE RETURN	Ready the fresh diskette on drive B and type a return when ready.

If the user responds with "A" rather than "B" above, the system will be written to drive A rather than B. SYSGEN will continue to type the prompt

```
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)
```

until the operator responds with a single carriage return, which stops the SYSGEN program with a system reboot.

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

The user can then go through the reboot process with the old or new diskette. Instead of performing the SYSGEN operation, the user can type

```
SAVE 34 CPMxx.COM
```

at the completion of the MOVCPM function, where "xx" is the value indicated in the SYSGEN message. The CP/M memory image on the currently logged disk is in a form that can be "patched." This is necessary when operating in a nonstandard environment where the BIOS must be altered for a particular peripheral device configuration, as described in Chapter 6.

Valid MOVCPM commands are

MOVCPM 48	Construct a 48K version of CP/M and start execution.
MOVCPM 48 *	Construct a 48K version of CP/M in preparation for permanent recording; response is READY FOR 'SYSGEN' OR 'SAVE 34 CPM48.COM'
MOVCPM * *	Construct a maximum memory version of CP/M and start execution.

The newly created system is serialized with the number attached to the original diskette and is subject to the conditions of the Digital Research Software Licensing Agreement.

1.7 BDOS Error Messages

There are three error situations that the Basic Disk Operating System intercepts during file processing. When one of these conditions is detected, the BDOS prints the message:

```
BDOS ERR ON d: error
```

where d is the drive name and "error" is one of the three error messages:

```
BAD SECTOR
SELECT
READ ONLY
```

The "BAD SECTOR" message indicates that the disk controller electronics has detected an error condition in reading or writing the diskette. This condition is generally caused by a malfunctioning disk controller or an extremely worn diskette. If the user finds that the CP/M reports this error more than once a month, the state of the controller electronics and the condition of the media should be checked. The user can also encounter this condition in reading files generated by a controller produced by a different manufacturer. Even though controllers are claimed to be IBM-compatible, one often finds small differences in recording formats. The MDS-800 controller, for example, requires two bytes of one's following the data CRC byte, which is not required in the IBM format. As a result, diskettes generated by the Intel MDS can be read by almost all other IBM-compatible systems, while disk files generated on other manufacturers' equipment will produce the "BAD SECTOR" message when read by the MDS. Recovery from this condition is accomplished by typing a ctl-C to reboot (the safest course), or a return,

which ignores the bad sector in the file operation. The user should, however, note that typing a return may destroy diskette integrity if the operation is a directory write. The user should be sure to have adequate backups in this case.

The "SELECT" error occurs when there is an attempt to address a drive beyond the range supported by the BIOS. In this case, the value of d in the error message gives the selected drive. The system reboots following any input from the console.

The "READ ONLY" message occurs when there is an attempt to write to a diskette or file that has been designated as read only in a STAT command or has been set to read only by the BDOS. The operator should reboot CP/M by using the warm start procedure (ctl-C) or by performing a cold start whenever the diskettes are changed. If a changed diskette is to be read but not written, BDOS allows the diskette to be changed without the warm or cold start, but internally marks the drive as read only. The status of the drive is subsequently changed to read/write if a warm or cold start occurs. On issuing this message, CP/M waits for input from the console. An automatic warm start takes place following any input.

1.8 Operation of CP/M on the MDS

This section gives operating procedures for using CP/M on the Intel MDS microcomputer development system. Basic knowledge of the MDS hardware and software systems is assumed.

CP/M is initiated in essentially the same manner as Intel's ISIS operating system. The disk drives are labeled 0 through 3 on the MDS, corresponding to CP/M drives A through D, respectively. The CP/M system diskette is inserted into drive 0, and the BOOT and RESET switches are depressed in sequence. The interrupt 2 light should go on at this point. The space bar is then depressed on the system console, and the light should go out (if it does not, the user should check connections and baud rates). The BOOT switch is turned off, and the CP/M sign-on message should appear at the selected console device, followed by the "A>" system prompt. The user can then issue the various resident and transient commands.

The CP/M system can be restarted (warm start) at any time by pushing the INT 0 switch on the front panel. The built-in Intel ROM monitor can be initiated by pushing the INT 7 switch (which generates an RST 7), except when operating under DDT, in which case the DDT program gets control instead.

Diskettes can be removed from the drives at any time, and the system can be shut down during operation without affecting data integrity. The user must not remove a diskette and replace it with another without rebooting the system (cold or warm start) unless the inserted diskette is "read only."

As a result of hardware hang-ups or malfunctions, CP/M may type the message

```
BDOS ERR ON d: BAD SECTOR
```

where d is the drive that has a permanent error. This error can occur when drive doors are opened and closed randomly, followed by disk operations, or can be caused by a diskette, drive, or controller failure. The user can optionally elect to ignore the error by typing a single return at the console. The error may produce a bad data record, requiring reinitialization of up to 128 bytes of data. The operator can reboot the CP/M system and try the operation again.

Termination of a CP/M session requires no special action, except that it is necessary to remove the diskettes before turning the power off to avoid random transients that often make their way to the drive electronics.

Factory-fresh, IBM-compatible diskettes should be used rather than diskettes that have previously been used with any ISIS version. In particular, the ISIS "FORMAT" operation produces nonstandard sector numbering throughout the diskette. This nonstandard numbering seriously degrades the performance of CP/M and will operate noticeably slower than the distribution version. If it becomes necessary to reformat a diskette (which should not be the case for standard diskettes), a program can be written under CP/M that causes the MDS 800 controller to reformat with sequential sector numbering (1-26) on each track.

IBM-compatible 8-inch diskettes in general do not need to be formatted. However, 5¼-inch diskettes will need to be formatted.

2 ED

2.1 Introduction to ED

ED is the context editor for CP/M, and is used to create and alter CP/M source files. ED is initiated in CP/M by typing

```
ED filename
```

or

```
ED filename.typ
```

In general, ED reads segments of the source file given by filename or filename.typ into the central memory, where the file is manipulated by the operator and subsequently written back to disk after alterations. If the source file does not exist before editing, it is created by ED and initialized to empty. The overall operation of ED is shown in Figure 2.1.

2.1.1 ED Operation

ED operates upon the source file, denoted in Figure 2.1 by x.y, and passes all text through a memory buffer where the text can be viewed or altered (the number of lines that can be maintained in the memory buffer varies with the line length, but has a total capacity of about 5000 characters in a 20K CP/M system). Text material that has been edited is written into a temporary work file under command of the operator. Upon termination of the edit, the memory buffer is written to the temporary file, followed by any remaining (unread) text in the source file. The name of the original file is changed from x.y to x.BAK so that the most recent previously edited source file can be reclaimed if necessary (see the CP/M commands ERASE and RENAME). The temporary file is then changed from x.$$$ to x.y, which becomes the resulting edited file.

The memory buffer is logically between the source file and working file as shown in Figure 2.2.

Figure 2.1 **Overall ED Operation**

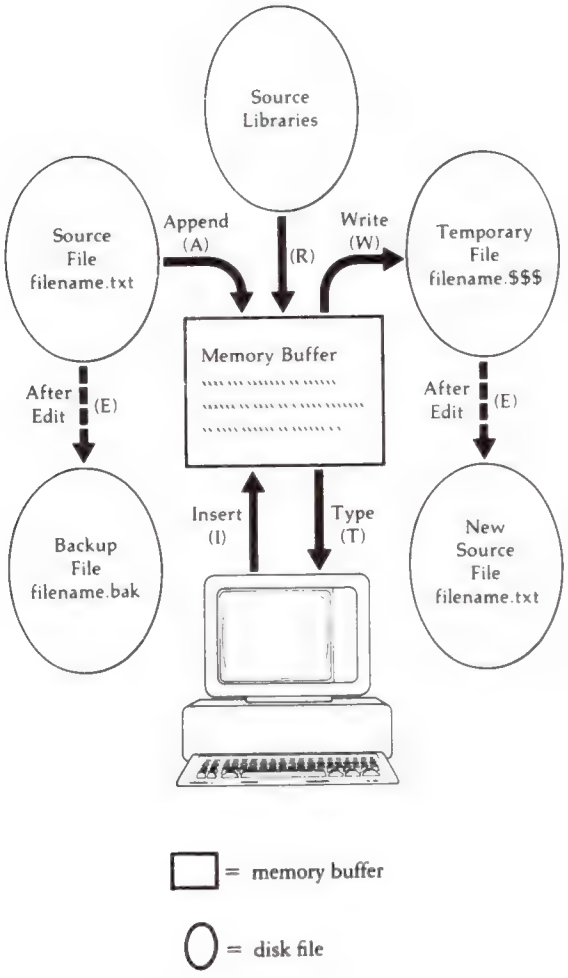
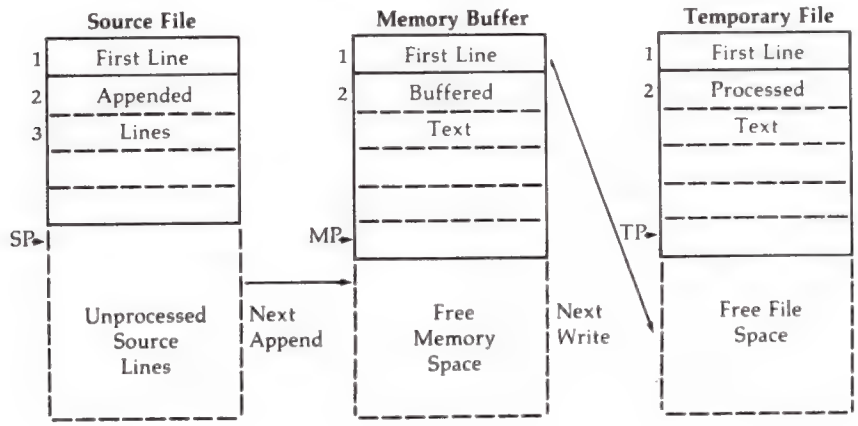


Figure 2.2 **Memory Buffer Organization**



SP = Source Pointer
MP = Memory Pointer
TP = Temporary Pointer

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

2.1.2 Text Transfer Functions

Given that n is an integer value in the range 0 through 65535, several single letter ED commands transfer lines of text from the source file through the memory buffer to the temporary (and eventually final) file. Single letter commands are shown in upper case, but can be typed in either upper or lower case.

nA	Append the next n unprocessed source lines from the source file at SP to the end of the memory buffer at MP. Increment SP and MP by n. If upper case translation is set (see the U command) and the A command is typed in upper case, all input lines will automatically be translated to upper case.
nW	Write the first n lines of the memory buffer to the temporary file free space. Shift the remaining lines n+1 through MP to the top of the memory buffer. Increment TP by n.
E	End the edit. Copy all buffered text to temporary file and copy all unprocessed source lines to temporary file. Rename files as described previously.
H	Move to head of new file by performing automatic E command. Temporary file becomes the new source file, the memory buffer is emptied, and a new temporary file is created (equivalent to issuing an E command, followed by a reinvocation of ED using x.y as the file to edit).
O	Return to original file. The memory buffer is emptied, the temporary file is deleted, and the SP is returned to position 1 of the source file. The effects of the previous editing commands are thus nullified.
Q	Quit edit with no file alterations, return to CP/M.

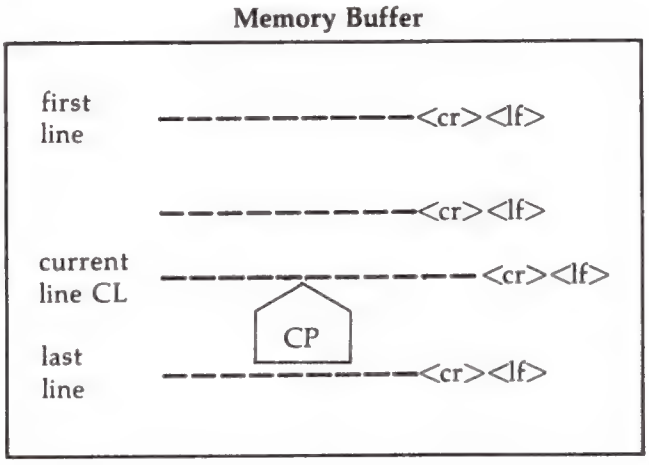
There are a number of special cases to consider. If the integer n is omitted in any ED command where an integer is allowed, then 1 is assumed. Thus, the commands A and W append one line and write one line, respectively. In addition, if a pound sign (#) is given in the place of n, then the integer 65535 is assumed (the largest value for n that is allowed). Since most reasonably sized source files can be contained entirely in the memory buffer, the command #A is often issued at the beginning of the edit to read the entire source file to memory. Similarly, the command #W writes the entire buffer to the temporary file. Two special forms of the A and W commands are provided as a convenience. The command 0A fills the current memory buffer at least half full, while 0W writes lines until the buffer is at least half empty. An error is issued if the memory buffer size is exceeded. The operator can then enter any command (such as W) that does not increase memory requirements. The remainder of any partial line read during the overflow will be brought into memory on the next successful append.

2.1.3 Memory Buffer Organization

The memory buffer can be considered a sequence of source lines brought in with the A command from a source file. The memory buffer has an associated (imaginary) character pointer CP that moves throughout the memory buffer under command of the operator. The memory buffer appears logically as shown in Figure 2.3 where the dashes represent characters of the source line of indefinite length, terminated by carriage-return (<cr>) and line-feed (<lf>) characters, and CP represents the imaginary character pointer. The

user should note that the CP is always located ahead of the first character of the first line, behind the last character of the last line, or between two characters. The current line CL is the source line that contains the CP.

Figure 2.3 **Logical Organization of Memory Buffer**



2.1.4 Line Numbers and ED Start-up

ED produces absolute line number prefixes that can be used to reference a line, or range of lines. The absolute line number is displayed at the beginning of each line when ED is in "insert mode" (see the I command in Section 2.1.5), where each line number takes the form

```
nnnnn:
```

where nnnnn is an absolute line number in the range of 1 to 65535. If the memory buffer is empty or if the current line is at the end of the memory buffer, nnnnn appears as 5 blanks.

The user may reference an absolute line number by preceding any command by a number followed by a colon, in the same format as the line number display. In this case, the ED program moves the current line reference to the absolute line number, if the line exists in the current memory buffer. The line denoted by the absolute line number must be in the memory buffer (see the A command). Thus, the command

```
345:T
```

is interpreted as "move to absolute 345, and type the line." Absolute line numbers are produced only during the editing process and are not recorded with the file. In particular, the line numbers will change following a deleted or expanded section of text.

The user may also reference an absolute line number as a backward or forward distance from the current line by preceding the absolute number by a colon. Thus, the command

```
:400T
```


ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

is interpreted as "type from the current line number through the line whose absolute number is 400." Combining the two line reference forms, the command

```
345::400T
```

for example, is interpreted as "move to absolute line 345, then type through absolute line 400." Absolute line references of this sort can precede any of the standard ED commands.

Line numbering is controlled by the "V" (Verify line numbers) command. Line numbering can be disabled by typing the "-V" command.

If the file to edit does not exist, ED types the message

```
NEW FILE
```

The user must enter an "i" command so that text can be inserted into the memory buffer by typing input lines terminated by carriage-returns. A single ctl-Z character returns ED to command mode.

2.1.5 Memory Buffer Operation

When ED begins, the memory buffer is empty. The operator may either append lines (A command) from the source file or enter the lines directly from the console with the insert command

```
I
```

ED then accepts any number of input lines, where each line terminates with a <cr> (the <lf> is supplied automatically), until a control-z (denoted by ↑z) is typed by the operator. The CP is positioned after the last character entered. The sequence

```
I<cr>
NOW IS THE<cr>
TIME FOR<cr>
ALL GOOD MEN<cr>
↑z
```

leaves the memory buffer as

```
NOW IS THE<cr><lf>
TIME FOR<cr><lf>
ALL GOOD MEN<cr><lf>
```

Generally, ED accepts command letters in upper or lower case. If the command is upper case, all input values associated with the command are translated to upper case. In particular, if the "I" command is typed, all input lines are automatically translated internally to upper case. The lower case form of the "i" command is most often used to allow both upper and lower case letters to be entered.

Various commands can be issued that manipulate the CP or display source text in the vicinity of the CP. The commands shown below with a preceding n indicate that an

optional unsigned value can be specified. When preceded by ±, the command can be unsigned, or have an optional preceding plus or minus sign. As before, the pound sign (#) is replaced by 65535. If an integer n is optional, but not supplied, then n = 1 is assumed. Finally, if a plus sign is optional, but none is specified, then + is assumed.

±B	Move CP to beginning of memory buffer if + and to bottom if -
±nC	Move CP by ±n characters (moving ahead if +), counting the <cr><lf> as two distinct characters.
±nD	Delete n characters ahead of CP if plus and behind CP if minus.
±nK	Kill (i.e., remove) ±n lines of source text using CP as the current reference. If CP is not at the beginning of the current line when K is issued, the characters before CP remain if + is specified, while the characters after CP remain if - is given in the command.
±nL	If n = 0, move CP to the beginning of the current line (if it is not already there). If n ≠ 0, first move the CP to the beginning of the current line and then move it to the beginning of the line that is n lines down (if +) or up (if -). The CP will stop at the top or bottom of the memory buffer if too large a value of n is specified.
±nT	If n = 0, type the contents of the current line up to CP. If n = 1, type the contents of the current line from CP to the end of the line. If n>1, type the current line along with n - 1 lines that follow, if + is specified. Similarly, if n>1 and - is given, type the previous n lines up to the CP. Any key can be depressed to abort long type-outs.
±n	Equivalent to ±nLT, which moves up or down and types a single line.

2.1.6 Command Strings

Any number of commands can be typed contiguously (up to the capacity of the console buffer) and are executed only after the <cr> is typed. Thus, the operator may use the CP/M console line editing operation to manipulate the input command line:

ctl-C	CP/M system reboot when typed at start of line.
ctl-E	Physical end of line: carriage is returned, but line is not sent until the carriage return key is depressed.
ctl-H	Backspace one character position.
ctl-J	Terminate current input (line feed).
ctl-M	Terminate current input (carriage return).
ctl-R	Retype current command line: types a "clean line" following character deletion with rubouts.
ctl-U	Delete the entire line typed at the console.
ctl-X	Same as ctl-U.
ctl-Z	End input from the console (used in PIP and ED).
rub/del	Delete and echo the last character typed at the console.

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

Suppose the memory buffer contains the characters shown in the previous section, with the CP following the last character of the buffer. The command strings shown below produce the results shown to the right. Use lower case command letters to avoid automatic translation of strings to upper case.

Command String	Effect	Resulting Memory Buffer
1. B2T<cr>	Move to beginning of buffer and type 2 lines: 'NOW IS THE TIME FOR'	NOW IS THE<cr><lf> TIME FOR<cr><lf> ALL GOOD MEN<cr><lf> CP
2. 5C0T<cr>	Move CP 5 characters and type the beginning of the line 'NOW I'	NOW I S THE <cr><lf> CP
3. 2L-T<cr>	Move two lines down and type previous line 'TIME FOR'	NOW IS THE<cr><lf> TIME FOR<cr><lf> ALL GOOD MEN<cr><lf> CP
4. -L#K<cr>	Move up one line, delete 65535 lines that follow	NOW IS THE<cr><lf> CP
5. I<cr> TIME TO<cr> INSERT<cr> ↑z	Insert two lines of text with automatic translation to upper case	NOW IS THE<cr><lf> TIME TO<cr><lf> INSERT<cr><lf> CP
6. -2L#T<cr>	Move up two lines and type 65535 lines ahead of CP 'NOW IS THE'	NOW IS THE<cr><lf> TIME TO<cr><lf> INSERT<cr><lf> CP
7. <cr>	Move down one line and type one line 'INSERT'	NOW IS THE<cr><lf> TIME TO<cr><lf> INSERT<cr><lf> CP

2.1.7 Text Search and Alteration

ED also has a command that locates strings within the memory buffer. The command takes the form

```
nF s  <cr>
```

or

```
nF s  ↑z
```

where s represents the string to match, followed by either a <cr> or ctl-Z, denoted by ↑z. ED starts at the current position of CP and attempts to match the string. The match is attempted n times, and if successful, the CP is moved directly after the string. If the n matches are not successful, the CP is not moved from its initial position. Search strings can include ctl-L, which is replaced by the pair of symbols <cr><lf>.

The following commands illustrate the use of the F command:

Command String	Effect	Resulting Memory Buffer
1. B#T<cr>	Move to beginning and type entire buffer	NOW IS THE<cr><lf> CP TIME FOR<cr><lf> ALL GOOD MEN<cr><lf>
2. FS T<cr>	Find the end of the string 'S T'	NOW IS T HE<cr><lf> CP
3. FI↑z0TT	Find the next 'I' and type to the CP; then type the remainder of the current line: 'ME FOR'	NOW IS THE<cr><lf> TI ME FOR<cr><lf> CP ALL GOOD MEN<cr><lf>

An abbreviated form of the insert command is also allowed, which is often used in conjunction with the F command to make simple textual changes. The form is

I s ↑z

or

I s<cr>

where s is the string to insert. If the insertion string is terminated by a ↑z, the string is inserted directly following the CP, and the CP positioned directly after the string. The action is the same if the command is followed by a <cr> except that a <cr><lf> is automatically inserted into the text following the string. Consider the following command sequences as examples of the F and I commands:

	Command String	Effect	Resulting Memory Buffer
1.	BITHIS IS ↑z<cr>	Insert 'THIS IS' at the beginning of the text	THIS IS NOW THE<cr><lf> CP TIME FOR<cr><lf> ALL GOOD MEN<cr><lf>
2.	FTIME↑z-4DIPLACE↑z<cr>	Find 'TIME' and delete it; then insert 'PLACE'	THIS IS NOW THE<cr><lf> PLACE FOR<cr><lf> CP ALL GOOD MEN<cr><lf>
3.	3FO↑z-3D5D1 CHANGES↑z<cr>	Find third occurrence of 'O' (i.e., the second 'O' in GOOD), delete previous 3 characters and the subsequent 5 characters; then insert 'CHANGES'	THIS IS NOW THE<cr><lf> PLACE FOR<cr><lf> ALL CHANGES<cr><lf> CP
4.	-8CISOURCE<cr>	Move back 8 characters and insert the line 'SOURCE<cr><lf>'	THIS IS NOW THE<cr><lf> PLACE FOR<cr><lf> ALL SOURCE<cr><lf> CP CHANGES<cr><lf>

ED also provides a single command that combines the F and I commands to perform simple string substitutions. The command takes the form

n S s_1↑z s_2 <cr>

or

n S s_1↑z s_2 ↑z

and has exactly the same effect as applying the following command string a total of n times:

F s_1↑z-kDIs_2 <cr>

or

F s_1↑z-kDIs_2 ↑z

where k is the length of the string. That is, ED searches the memory buffer starting at the current position of CP and successively substitutes the second string for the first string until the end of buffer or until the substitution has been performed n times.

As a convenience, a command similar to F is provided by ED, which automatically appends and writes lines as the search proceeds. The form is

n N s <cr>

or

n N s ↑z

which searches the entire source file for the nth occurrence of the strings (the user should recall that F fails if the string cannot be found in the current buffer). The operation of the N command is precisely the same as F except in the case that the string cannot be found within the current memory buffer. In this case, the entire memory content is written (i.e., an automatic #W is issued). Input lines are then read until the buffer is at least half full or the entire source file is exhausted. The search continues in this manner until the string has been found n times or until the source file has been completely transferred to the temporary file.

A final line editing function, called the juxtaposition command, takes the form

n J s_1↑z s_2↑z s_3 <cr>

or

n J s_1↑z s_2↑z s_3 ↑z

with the following action applied n times to the memory buffer: search from the current CP for the next occurrence of the string s_1. If found, insert the string s_2, and move CP to follow s_2. Then delete all characters following CP up to (but not including) the string s_3, leaving CP directly after s_2. If s_3 cannot be found, then no deletion is made. If the current line is

NOW IS THE TIME<cr><lf>

the command

JW ↑zWHAT↑z↑l<cr>

results in

NOW WHAT <cr <lf>

(The user should recall that ↑l (ctl-L) represents the pair <cr><lf> in search and substitute strings.)

The number of characters allowed by ED in the F, S, N, and J commands is limited to 100 symbols.

2.1.8 Source Libraries

ED also allows the inclusion of source libraries during the editing process with the R command. The form of this command is

```
R filename ↑z
```

or

```
R filename <cr>
```

where filename is the primary filename of a source file on the disk with an assumed filetype of 'LIB'. ED reads the specified file, and places the characters into the memory buffer after CP, in a manner similar to the I command. Thus, if the command

```
RMACRO<cr>
```

is issued by the operator, ED reads from the file MACRO.LIB until the end-of-file and automatically inserts the characters into the memory buffer.

ED also includes a "block move" facility implemented through the X (Xfer) command. The form

```
nX
```

transfers the next n lines from the current line to a temporary file called

```
X$$$$$$.LIB
```

which is active only during the editing process. In general, the user can reposition the current line reference to any portion of the source file and transfer lines to the temporary file. The transferred lines accumulate one after another in this file and can be retrieved by simply typing

```
R
```

which is the trivial case of the library read command. In this case, the entire transferred set of lines is read into the memory buffer. The user should note that the X command does not remove the transferred lines from the memory buffer, although a K command can be used directly after the X, and the R command does not empty the transferred LIB file. That is, given that a set of lines has been transferred with the X command, they can be reread any number of times back into the source file. The command

```
0X
```

is provided, however, to empty the transferred line file.

The user should note that upon normal completion of the ED program through Q or E, the temporary LIB file is removed. If ED is aborted through ctl-C, the LIB file will exist if lines have been transferred, but will generally be empty (a subsequent ED invocation will erase the temporary file).

2.1.9 Repetitive Command Execution

The macro command M allows the ED user to group ED commands together for repeated evaluation. The M command takes the form

```
n M CS  <cr>
```

or

```
n M CS  ↑z
```

where CS represents a string of ED commands, not including another M command. ED executes the command string n times if n>1. If n=0 or 1, the command string is executed repetitively until an error condition is encountered (e.g., the end of the memory buffer is reached with an F command).

As an example, the following macro changes all occurrences of GAMMA to DELTA within the current buffer, and types each line that is changed

```
MFGAMMA↑z-5DIDELTA↑z0TT<cr>
```

or equivalently

```
MSGAMMA↑zDELTA↑z0TT<cr>
```

2.2 ED Error Conditions

On error conditions, ED prints the message "BREAK X AT C" where X is one of the error indicators shown below:

?	Unrecognized command.
>	Memory buffer full (use one of the commands D, K, N, S, or W to remove characters); F, N, or S strings too long.
#	Cannot apply command the number of times specified (e.g., in F command).
O	Cannot open LIB file in R command.

If there is a disk error, CP/M displays the following message:

```
BDOS ERR on d: BAD SECTOR
```

The operator can choose to ignore the error by pressing the return key at the console (in this case, the memory buffer data should be examined to see if they were incorrectly read), or the user can reset the system by ctl-C and reclaim the backup file if its exists. The file can be reclaimed by first typing the contents of the BAK file to ensure that it contains the proper information

```
TYPE x.BAK
```

where x is the file being edited. Then remove the primary file

```
ERA x.y
```

and rename the BAK file

```
REN x.y=x.BAK
```

The file can then be reedited, starting with the previous version.

ED also takes file attributes into account. If the operator attempts to edit a read/only file, the message

```
** FILE IS READ/ONLY **
```

appears at the console. The file can be loaded and examined, but cannot be altered. Normally the operator simply ends the edit session and uses STAT to change the file attribute to R/W. If the edited file has the "system" attribute set, the message

```
'SYSTEM' FILE NOT ACCESSIBLE
```

is displayed and the edit session is aborted. Again, the STAT program can be used to change the system attribute, if desired.

2.3 Control Characters and Commands

The following tabulation summarizes the control characters and commands available in ED:

Control Character	Function
ctl-C	System reboot
ctl-E	Physical <cr><lf> (not actually entered in command)
ctl-H	Backspace
ctl-J	Logical tab (cols 1, 9, 16,...)
ctl-L	Logical <cr><lf> in search and substitute strings
ctl-R	Repeat line
ctl-U	Line delete
ctl-X	Line delete
ctl-Z	String terminator
rub/del	Character delete

Command	Function
nA	Append lines
±B	Begin or bottom of buffer
±nC	Move character positions
±nD	Delete characters
E	End edit and close files (normal end)
nF	Find string
H	End edit, close and reopen files
I	Insert characters, use i if both upper and lower case characters are to be entered
nJ	Place strings in juxtaposition
±nK	Kill lines
±nL	Move down/up lines
nM	Macro definition
nN	Find next occurrence with autoscan

O	Return to original file
±nP	Move and print pages
Q	Quit with no file changes
R	Read library file
nS	Substitute strings
±nT	Type lines
±U	Translate lower to upper case if U, no translation if -U
±V	Verify line numbers, or show remaining free character space
0V	A special case of the V command, 0V, prints the memory buffer statistics in the form free/total where free is the number of free bytes in the memory buffer (in decimal) and total is the size of the memory buffer
nW	Write lines
nZ	Wait (sleep) for approximately n seconds
±n	Move and type (±nLT).

Because of common typographical errors, ED requires several potentially disastrous commands to be typed as single letters, rather than in composite commands. The commands

E(end), H(head), O(original), Q(quit)

must be typed as single letter commands.

The commands I, J, M, N, R, and S should be typed as i, j, m, n, r, and s if both upper and lower case characters are used in the operation, otherwise all characters are converted to upper case. When a command is entered in upper case, ED automatically converts the associated string to upper case, and vice-versa.

3

CP/M Assembler

3.1 Introduction

The CP/M assembler reads assembly language source files from the diskette and produces 8080 machine language in Intel hex format. The CP/M assembler is initiated by typing

```
ASM filename
```

or

```
ASM filename.parms
```

In both cases, the assembler assumes there is a file on the diskette with the name

```
filename.ASM
```

which contains an 8080 assembly language source file. The first and second forms shown above differ only in that the second form allows parameters to be passed to the assembler to control source file access and hex and print file destinations.

In either case, the CP/M assembler loads and prints the message

```
CP/M ASSEMBLER VER n.n
```

where n.n is the current version number. In the case of the first command, the assembler reads the source file with assumed file type ASM and creates two output files

```
filename.HEX
```

and

```
filename.PRN
```

The HEX file contains the machine code corresponding to the original program in Intel hex format, and the PRN file contains an annotated listing showing generated machine

code, error flags, and source lines. If errors occur during translation, they will be listed in the PRN file as well as at the console.

The form ASM filename parms can be used to redirect input and output files from their defaults. In this case, the parms portion of the command is a three-letter group that specifies the origin of the source file, the destination of the hex file, and the destination of the print file. The form is

```
filename.p1p2p3
```

where p1, p2, and p3 are single letters

P1: A,B, ..., P designates the disk name that contains the source file

p2: A,B, ..., P designates the disk name that will receive the hex file

Z skips the generation of the hex file

p3: A,B, ..., P designates the disk name that will receive the print file

X places the listing at the console

Z skips generation of the print file

Thus, the command

```
ASM X.AAA
```

indicates that the source file (X.ASM) is to be taken from disk A and that the hex (X.HEX) and print (X.PRN) files are also to be created on disk A. This form of the command is implied if the assembler is run from disk A. That is, given that the operator is currently addressing disk A, the above command is equivalent to

```
ASM X
```

The command

```
ASM X.ABX
```

indicates that the source file is to be taken from disk A, the hex file is to be placed on disk B, and the listing file is to be sent to the console. The command

```
ASM X.BZZ
```

takes the source file from disk B and skips the generation of the hex and print files (this command is useful for fast execution of the assembler to check program syntax).

The source program format is compatible with the Intel 8080 assembler (macros are not implemented in ASM; see the optional MAC macro assembler). There are certain extensions in the CP/M assembler that make it somewhat easier to use. These extensions are described below.

3.2 Program Format

An assembly language program acceptable as input to the assembler consists of a sequence of statements of the form

```
line# label operation operand ;comment
```

where any or all of the fields may be present in a particular instance. Each assembly language statement is terminated with a carriage return and line feed (the line feed is inserted automatically by the ED program), or with the character !, which is treated as an end-of-line by the assembler (thus, multiple assembly language statements can be written on the same physical line if separated by exclamation mark symbols).

The line# is an optional decimal integer value representing the source program line number, and ASM ignores this field if present.

The label field takes the form

```
identifier
```

or

```
identifier:
```

and is optional, except where noted in particular statement types. The identifier is a sequence of alphanumeric characters where the first character is alphabetic. Identifiers can be freely used by the programmer to label elements such as program steps and assembler directives, but cannot exceed 16 characters in length. All characters are significant in an identifier, except for the embedded dollar symbol ($), which can be used to improve readability of the name. Further, all lower case alphabetics are treated as if they were upper case. The following are all valid instances of labels

```
x           xy          long$name
x:          yxl:        longer$named$data:
X1Y2        X1x2        x234$5678$9012$3456:
```

The operation field contains either an assembler directive or pseudo-operation, or an 8080 machine operation code. The pseudo-operations and machine operation codes are described below.

The operand field of the statement, in general, contains an expression formed out of constants and labels, along with arithmetic and logical operations on these elements. Again, the complete details of properly formed expressions are given below.

The comment field contains arbitrary characters following the ; symbol until the next real or logical end-of-line. These characters are read, listed, and otherwise ignored by the assembler. The CP/M assembler also treats statements that begin with an * in column one as comment statements, which are listed and ignored in the assembly process.

The assembly language program is formulated as a sequence of statements of the above form, terminated by an optional END statement. All statements following the END are ignored by the assembler.

3.3 Forming the Operand

To describe the operation codes and pseudo-operations completely, it is necessary first to present the form of the operand field, since it is used in nearly all statements. Expressions in the operand field consist of simple operands (labels, constants, and reserved words), combined in properly formed subexpressions by arithmetic and logical operators. The expression computation is carried out by the assembler as the assembly proceeds. Each expression must produce a 16-bit value during the assembly. Further, the number of significant digits in the result must not exceed the intended use. That is, if an expression is to be used in a byte move immediate instruction, the most significant 8 bits of the expression must be zero. The restriction on the expression significance is given with the individual instructions.

3.3.1 Labels

As discussed above, a label is an identifier that occurs on a particular statement. In general, the label is given a value determined by the type of statement that it precedes. If the label occurs on a statement that generates machine code or reserves memory space (e.g., a MOV instruction or a DS pseudo-operation), the label is given the value of the program address that it labels. If the label precedes an EQU or SET, the label is given the value that results from evaluating the operand field. Except for the SET statement, an identifier can label only one statement.

When a label appears in the operand field, its value is substituted by the assembler. This value can then be combined with other operands and operators to form the operand field for a particular instruction.

3.3.2 Numeric Constants

A numeric constant is a 16-bit value in one of several bases. The base, called the radix of the constant, is denoted by a trailing radix indicator. The radix indicators are

B	binary constant (base 2)
O	octal constant (base 8)
Q	octal constant (base 8)
D	decimal constant (base 10)
H	hexadecimal constant (base 16)

Q is an alternate radix indicator for octal numbers since the letter O is easily confused with the digit 0. Any numeric constant that does not terminate with a radix indicator is assumed to be a decimal constant.

A constant is thus composed as a sequence of digits, followed by an optional radix indicator, where the digits are in the appropriate range for the radix. That is, binary constants must be composed of 0 and 1 digits, octal constants can contain digits in the range 0-7, while decimal constants contain decimal digits. Hexadecimal constants contain decimal digits as well as hexadecimal digits A (10D), B (11D), C (12D), D (13D), E (14D), and F (15D). The user should note that the leading digit of a hexadecimal constant must be a decimal digit to avoid confusing a hexadecimal constant with an identifier (a leading 0 will always suffice). A constant composed in this manner must evaluate to a binary number that can be contained within a 16-bit counter, otherwise it is truncated on the right by the assembler. Similar to identifiers, imbedded $ signs are allowed within constants to improve their readability. Finally, the radix indicator is translated to upper case if a lower case letter is encountered. The following are all valid instances of numeric constants

```
1234        1234D     1100B     1111$0000$1111$0000B
1234H       0FFEH     3377O     33$77$22Q
3377o       0fe3h     1234d     0ffffh
```

3.3.3 Reserved Words

There are several reserved character sequences that have predefined meanings in the

operand field of a statement. The names of 8080 registers are given below. When they are encountered, they produce the values shown to the right.

A	7
B	0
C	1
D	2
E	3
H	4
L	5
M	6
SP	6
PSW	6

(Again, lower case names have the same values as their upper case equivalents.) Machine instructions can also be used in the operand field and evaluate to their internal codes. In the case of instructions that require operands, where the specific operand becomes a part of the binary bit pattern of the instruction (e.g., MOV A,B), the value of the instruction (in this case MOV) is the bit pattern of the instruction with zeroes in the optional fields (e.g., MOV produces 40H).

When the symbol $ occurs in the operand field (not imbedded within identifiers and numeric constants), its value becomes the address of the next instruction to generate, not including the instruction contained within the current logical line.

3.3.4 String Constants

String constants represent sequences of ASCII characters and are represented by enclosing the characters within apostrophe symbols ('). All strings must be fully contained within the current physical line (thus allowing ! symbols within strings) and must not exceed 64 characters in length. The apostrophe character itself can be included within a string by representing it as a double apostrophe (the two keystrokes ''), which becomes a single apostrophe when read by the assembler. In most cases, the string length is restricted to either one or two characters (the DB pseudo-operation is an exception), in which case the string becomes an 8- or 16-bit value, respectively. Two character strings become a 16-bit constant, with the second character as the low order byte, and the first character as the high order byte.

The value of a character is its corresponding ASCII code. There is no case translation within strings, and thus both upper and lower case characters can be represented. The user should note, however, that only graphic (printing) ASCII characters are allowed within strings.

Valid strings are	which represent
'A' 'AB' 'ab' 'c'	A AB ab c
'' 'a''' '''' ''''	a' ' '
'Walla Walla Wash.'	Walla Walla Wash.
'She said ''Hello'' to me.'	She said ''Hello'' to me
'I said ''Hello'' to her.'	I said ''Hello'' to her

3.3.5 Arithmetic and Logical Operators

The operands described above can be combined in normal algebraic notation using any combination of properly formed operands, operators, and parenthesized expressions. The operators recognized in the operand field are

a + b	unsigned arithmetic sum of a and b
a - b	unsigned arithmetic difference between a and b
+ b	unary plus (produces b)
- b	unary minus (identical to 0 - b)
a * b	unsigned magnitude multiplication of a and b
a / b	unsigned magnitude division of a by b
a MOD b	remainder after a / b
NOT b	logical inverse of b (all 0s become 1s, 1s become 0s), where b is considered a 16-bit value
a AND b	bit-by-bit logical and of a and b
a OR b	bit-by-bit logical or of a and b
a XOR b	bit-by-bit logical exclusive or of a and b
a SHL b	the value that results from shifting a to the left by an amount b, with zero fill
a SHR b	the value that results from shifting a to the right by an amount b, with zero fill.

In each case, a and b represent simple operands (labels, numeric constants, reserved words, and one or two character strings) or fully enclosed parenthesized subexpressions such as

```
10+20   10h+37Q   LI /3   (L2+4) SHR 3
('a' and 5fh) + '0'   ('B'+B) OR (PSW+M)
(1+(2+c)) shr (A-(B+1))
```

Note that all computations are performed at assembly time as 16-bit unsigned operations. Thus, -1 is computed as 0-1, which results in the value 0ffffh (i.e., all1s). The resulting expression must fit the operation code in which it is used. For example, if the expression is used in an ADI (add immediate) instruction, the high order 8 bits of the expression must be zero. As a result, the operation ADI -1 produces an error message (-1 becomes 0ffffh, which cannot be represented as an 8-bit value), while ADI (-1) AND 0FFH is accepted by the assembler since the AND operation zeroes the high order bits of the expression.

3.3.6 Precedence of Operators

As a convenience to the programmer, ASM assumes that operators have a relative precedence of application that allows the programmer to write expressions without nested levels of parentheses. The resulting expression has assumed parentheses that are defined by the relative precedence. The order of application of operators in unparenthesized expressions is listed below. Operators listed first have highest precedence (they are applied first in an unparenthesized expression), while operators listed last have lowest

precedence. Operators listed on the same line have equal precedence, and are applied from left to right as they are encountered in an expression

```
* / MOD SHL SHR
- +
NOT
AND
OR XOR
```

Thus, the expressions shown to the left below are interpreted by the assembler as the fully parenthesized expressions shown to the right

a * b + c	(a * b) + c
a + b * c	a + (b * c)
a MOD b * c SHL d	((a MOD b) * c) SHL d
a OR b AND NOT c + d SHL e	a OR (b AND (NOT (c + (d SHL e))))

Balanced parenthesized subexpressions can always be used to override the assumed parentheses; thus, the last expression above could be rewritten to force application of operators in a different order as

```
(a OR b) AND (NOT c) + d SHL e
```

resulting in the assumed parentheses

```
(a OR b) AND ((NOT c) + (d SHL e))
```

An unparenthesized expression is well-formed only if the expression that results from inserting the assumed parentheses is well-formed.

3.4 Assembler Directives

Assembler directives are used to set labels to specific values during the assembly, perform conditional assembly, define storage areas, and specify starting addresses in the program. Each assembler directive is denoted by a pseudo-operation that appears in the operation field of the line. The acceptable pseudo-operations are

ORG	set the program or data origin
END	end program, optional start address
EQU	numeric "equate"
SET	numeric "set"
IF	begin conditional assembly
ENDIF	end of conditional assembly
DB	define data bytes

DW	define data words
DS	define data storage area

The individual directives are detailed below.

3.4.1 The ORG Directive

The ORG statement takes the form

```
label  ORG  expression
```

where "label" is an optional program identifier and expression is a 16-bit expression, consisting of operands that are defined before the ORG statement. The assembler begins machine code generation at the location specified in the expression. There can be any number of ORG statements within a particular program, and there are no checks to ensure that the programmer is not defining overlapping memory areas. The user should note that most programs written for the CP/M system begin with an ORG statement of the form

```
ORG  100H
```

which causes machine code generation to begin at the base of the CP/M transient program area. If a label is specified in the ORG statement, the label is given the value of the expression (this label can then be used in the operand field of other statements to represent this expression).

3.4.2 The END Directive

The END statement is optional in an assembly language program, but if it is present it must be the last statement (all subsequent statements are ignored in the assembly). The two forms of the END directive are

```
label  END
label  END  expression
```

where the label is again optional. If the first form is used, the assembly process stops, and the default starting address of the program is taken as 0000. Otherwise, the expression is evaluated, and becomes the program starting address (this starting address is included in the last record of the Intel formatted machine code hex file, which results from the assembly). Thus, most CP/M assembly language programs end with the statement

```
END  100H
```

resulting in the default starting address of 100H (beginning of the transient program area).

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

3.4.3 The EQU Directive

The EQU (equate) statement is used to set up synonyms for particular numeric values. The form is

```
label   EQU   expression
```

where the label must be present and must not label any other statement. The assembler evaluates the expression, and assigns this value to the identifier given in the label field. The identifier is usually a name that describes the value in a more human-oriented manner. Further, this name is used throughout the program to "parameterize" certain functions. Suppose data received from a teletype appear on a particular input port and data are sent to the teletype through the next output port in sequence. The series of equate statements could be used to define these ports for a particular hardware environment

```
TTYBASE     EQU 10H          ;BASE PORT NUMBER FOR TTY
TTYIN       EQU TTYBASE      ;TTY DATA IN
TTYOUT      EQU TTYBASE+1    ;TTY DATA OUT
```

At a later point in the program, the statements that access the teletype could appear as

```
IN          TTYIN       ;READ TTY DATA TO REG-A
...
OUT         TTYOUT      ;WRITE DATA TO TTY FROM REG-A
```

making the program more readable than if the absolute I/O ports had been used. Further, if the hardware environment is redefined to start the teletype communications ports at 7FH instead of 10H, the first statement need only be changed to

```
TTYBASE   EQU   7FH   ;BASE PORT NUMBER FOR TTY
```

and the program can be reassembled without changing any other statements.

3.4.4 The SET Directive

The SET statement is similar to the EQU, taking the form

```
label   SET   expression
```

except that the label can occur on other SET statements within the program. The expression is evaluated and becomes the current value associated with the label. Thus, the EQU statement defines a label with a single value, while the SET statement defines a value that is valid from the current SET statement to the point where the label occurs on the next SET statement. The use of the SET is similar to the EQU statement, but is used most often in controlling conditional assembly.

3.4.5 The IF and ENDIF Directives

The IF and ENDIF statements define a range of assembly language statements that are to be included or excluded during the assembly process. The form is

```
IF   expression
statement#1
statement#2
  ...
statement#n
ENDIF
```

Upon encountering the IF statement, the assembler evaluates the expression following the IF (all operands in the expression must be defined ahead of the IF statement). If the expression evaluates to a nonzero value, then statement#l through statement#n are assembled; if the expression evaluates to zero, the statements are listed but not assembled. Conditional assembly is often used to write a single "generic" program that includes a number of possible run-time environments, with only a few specific portions of the program selected for any particular assembly. The following program segments, for example, might be part of a program that communicates with either a teletype or a CRT console (but not both) by selecting a particular value for TTY before the assembly begins.

```
TRUE      EQU     0FFFFH      ;DEFINE VALUE OF TRUE
FALSE     EQU     NOT TRUE    ;DEFINE VALUE OF FALSE
;
TTY       EQU     TRUE        ;TRUE IF TTY, FALSE IF CRT
;
TTYBASE   EQU     10H         ;BASE OF TTY I/O PORTS
CRTBASE   EQU     20H         ;BASE OF CRT I/O PORTS
          IF      TTY         ;ASSEMBLE RELATIVE TO
                              ;TTYBASE
CONIN     EQU     TTYBASE     ;CONSOLE INPUT
CONOUT    EQU     TTYBASE+1   ;CONSOLE OUTPUT
          ENDIF

;         IF      NOT TTY     ;ASSEMBLE RELATIVE TO
                              ;CRTBASE
CONIN     EQU     CRTBASE     ;CONSOLE INPUT
CONOUT    EQU     CRTBASE+1   ;CONSOLE OUTPUT

          ENDIF
          ...
          IN      CONIN       ;READ CONSOLE DATA
          ...
          OUT     CONOUT      ;WRITE CONSOLE DATA
```

In this case, the program would assemble for an environment where a teletype is connected, based at port 10H. The statement defining TTY could be changed to

```
TTY       EQU     FALSE
```

and, in this case, the program would assemble for a CRT based at port 20H.

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

3.4.6 The DB Directive

The DB directive allows the programmer to define initialized storage areas in single precision (byte) format. The statement form is

```
label   DB   e#1, e#2, ..., e#n
```

where e#1 through e#n are either expressions that evaluate to 8-bit values (the high order bit must be zero) or are ASCII strings of length no greater than 64 characters. There is no practical restriction on the number of expressions included on a single source line. The expressions are evaluated and placed sequentially into the machine code file following the last program address generated by the assembler. String characters are similarly placed into memory starting with the first character and ending with the last character. Strings of length greater than two characters cannot be used as operands in more complicated expressions. The user should note that ASCII characters are always placed in memory with the parity bit reset (0). Also, there is no translation from lower to upper case within strings. The optional label can be used to reference the data area throughout the remainder of the program. Examples of valid DB statements are

```
data:       DB      0,1,2,3,4,5
            DB      data and 0ffh,5,377Q,1+2+3+4

sign-on:    DB      'please type your name',cr,lf,0
            DB      'AB' SHR 8, 'C', 'DE' AND 7FH
```

3.4.7 The DW Directive

The DW statement is similar to the DB statement except double precision (two byte) words of storage are initialized. The form is

```
label       DW      e#1, e#2, ..., e#n
```

where e#1 through e#n are expressions that evaluate to 16-bit results. The user should note that ASCII strings of one or two characters are allowed, but strings longer than two characters are disallowed. In all cases, the data storage is consistent with the 8080 processor: the least significant byte of the expression is stored first in memory, followed by the most significant byte. Examples are

```
doub:       DW      0ffefh,doub+4,signon-$,255+255
            DW      'a', 5, 'ab', 'CD', 6 shl 8 or 11b.
```

3.4.8 The DS Directive

The DS statement is used to reserve an area of uninitialized memory, and takes the form

```
label       DS      expression
```

where the label is optional. The assembler begins subsequent code generation after the area reserved by the DS. Thus, the DS statement given above has exactly the same effect as the statement

```
label:  EQU  $  ;LABEL VALUE IS CURRENT CODE LOCATION
        ORG  $+expression  ;MOVE PAST RESERVED AREA
```

3.5 Operation Codes

Assembly language operation codes form the principal part of assembly language programs and form the operation field of the instruction. In general, ASM accepts all the standard mnemonics for the Intel 8080 microcomputer, which are given in detail in Intel's "8080 Assembly Language Programming Manual." Labels are optional on each input line. The individual operators are listed briefly in the following sections for completeness, although it is understood that the Intel manuals should be referenced for exact operator details. In the following tables,

e3	represents a 3-bit value in the range 0-7 which can be one of the predefined registers A, B, C, D, E, H, L, M, SP, or PSW.
e8	represents an 8-bit value in the range 0-255.
e16	represents a 16-bit value in the range 0-65535.

These expressions can be formed from an arbitrary combination of operands and operators. In some cases, the operands are restricted to particular values within the allowable range, such as the PUSH instruction. These cases will be noted as they are encountered.

In the sections that follow, each operation code is listed in its most general form, along with a specific example, with a short explanation and special restrictions.

3.5.1 Jumps, Calls, and Returns

The Jump, Call, and Return instructions allow several different forms that test the condition flags set in the 8080 microcomputer CPU. The forms are

JMP	e16	JMP L1	Jump unconditionally to label
JNZ	e16	JNZ L2	Jump on nonzero condition to label
JZ	e16	JZ 100H	Jump on zero condition to label
JNC	e16	JNC L1+4	Jump no carry to label
JC	e16	JC L3	Jump on carry to label
JPO	e16	JPO $+8	Jump on parity odd to label
JPE	e16	JPE L4	Jump on even parity to label
JP	e16	JP GAMMA	Jump on positive result to label
JM	e16	JM al	Jump on minus to label.
CALL	e16	CALL S1	Call subroutine unconditionally
CNZ	e16	CNZ S2	Call subroutine on nonzero condition

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

CZ	e16	CZ 100H	Call subroutine on zero condition
CNC	e16	CNC S1+4	Call subroutine if no carry set
CC	e16	CC S3	Call subroutine if carry set
CPO	e16	CPO $+8	Call subroutine if parity odd
CPE	e16	CPE S4	Call subroutine if parity even
CP	e16	CP GAMMA	Call subroutine if positive result
CM	e16	CM b1$c2	Call subroutine if minus flag.
RST	e3	RST 0	Programmed restart, equivalent to CALL 8*e3, except one byte call.
RET			Return from subroutine
RNZ			Return if nonzero flag set
RZ			Return if zero flag set
RNC			Return if no carry
RC			Return if carry flag set
RPO			Return if parity is odd
RPE			Return if parity is even
RP			Return if positive result
RM			Return if minus flag is set.

3.5.2 Immediate Operand Instructions

Several instructions are available that load single or double precision registers or single precision memory cells with constant values, along with instructions that perform immediate arithmetic or logical operations on the accumulator (register A).

MVI e3,e8	MVI B,255	Move immediate data to register A, B, C, D, E, H, L, or M (memory)
ADI e8	ADI 1	Add immediate operand to A without carry
ACI e8	ACI 0FFH	Add immediate operand to A with carry
SUI e8	SUI L + 3	Subtract from A without borrow (carry)
SBI e8	SBI L AND 11B	Subtract from A with borrow (carry)
ANI e8	ANI $ AND 7FH	Logical "and" A with immediate data
XRI e8	XRI 1111$0000B	"Exclusive or" A with immediate data
ORI e8	ORI L AND 1+1	Logical "or" A with immediate data

CPI e8	CPI 'a'	Compare A with immediate data (same as SUI except register A not changed).
LXI e3,e16	LXI B,100H	Load extended immediate to register pair (e3 must be equivalent to B,D,H, or SP).

3.5.3 Increment and Decrement Instructions

The 8080 provides instructions for incrementing or decrementing single and double precision registers. The instructions are

INR e3	INR E	Single precision increment register (e3 produces one of A, B, C, D, E, H, L, M)
DCR e3	DCR A	Single precision decrement register (e3 produces one of A, B, C, D, E, H, L, M)
INX e3	INX SP	Double precision increment register pair (e3 must be equivalent to B,D,H, or SP)
DCX e3	DCX B	Double precision decrement register pair (e3 must be equivalent to B,D,H, or SP).

3.5.4 Data Movement Instructions

Instructions that move data from memory to the CPU and from CPU to memory are given below.

MOV e3,e3	MOV A,B	Move data to leftmost element from rightmost element (e3 produces one of A,B,C,D,E,H,L, or M). MOV M,M is disallowed
LDAX e3	LDAX B	Load register A from computed address (e3 must produce either B or D)
STAX e3	STAX D	Store register A to computed address (e3 must produce either B or D)
LHLD e16	LHLD L1	Load HL direct from location e16 (double precision load to H and L)
SHLD e16	SHLD L5+x	Store HL direct to location e16 (double precision store from H and L to memory)
LDA e16	LDA Gamma	Load register A from address e16

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

STA e16	STA X3-5	Store register A into memory at e16
POP e3	POP PSW	Load register pair from stack, set SP (e3 must produce one of B, D, H, or PSW)
PUSH e3	PUSH B	Store register pair into stack, set SP (e3 must produce one of B, D, H, or PSW)
IN e8	IN 0	Load register A with data from port e8
OUT e8	OUT 255	Send data from register A to port e8
XTHL		Exchange data from top of stack with HL
PCHL		Fill program counter with data from HL
SPHL		Fill stack pointer with data from HL
XCHG		Exchange DE pair with HL pair

3.5.5 Arithmetic Logic Unit Operations

Instructions that act upon the single precision accumulator to perform arithmetic and logic operations are

ADD e3	ADD B	Add register given by e3 to accumulator without carry (e3 must produce one of A, B, C, D, E, H, or L)
ADC e3	ADC L	Add register to A with carry, e3 as above
SUB e3	SUB H	Subtract reg e3 from A without carry, e3 is defined as above
SBB e3	SBB 2	Subtract register e3 from A with carry, e3 defined as above
ANA e3	ANA 1+1	Logical "and" reg with A, e3 as above
XRA e3	XRA A	"Exclusive or" with A, e3 as above
ORA e3	ORA B	Logical "or" with A, e3 defined as above
CMP e3	CMP H	Compare register with A, e3 as above
DAA		Decimal adjust register A based upon last arithmetic logic unit operation
CMA		Complement the bits in register A

STC		Set the carry flag to 1
CMC		Complement the carry flag
RLC		Rotate bits left, (re)set carry as a side effect (high order A bit becomes carry)
RRC		Rotate bits right, (re)set carry as side effect (low order A bit becomes carry)
RAL		Rotate carry/A register to left (carry is involved in the rotate)
RAR		Rotate carry/A register to right (carry is involved in the rotate)
DAD e3	DAD B	Double precision add register pair e3 to HL (e3 must produce B, D, H, or SP).

3.5.6 Control Instructions

The four remaining instructions categorized as control instructions are

HLT	Halt the 8080 processor
DI	Disable the interrupt system
EI	Enable the interrupt system
NOP	No operation.

3.6 Error Messages

When errors occur within the assembly language program, they are listed as single character flags in the leftmost position of the source listing. The line in error is also echoed at the console so that the source listing need not be examined to determine if errors are present. The error codes are

D	Data error: element in data statement cannot be placed in the specified data area.
E	Expression error: expression is ill-formed and cannot be computed at assembly time.
L	Label error: label cannot appear in this context (may be duplicate label).
N	Not implemented: features that will appear in future ASM versions (e.g., macros) are recognized, but flagged in this version.
O	Overflow: expression is too complicated (i.e., too many pending operators) to be computed and should be simplified.
P	Phase error: label does not have the same value on two subsequent passes through the program.

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

R	Register error: the value specified as a register is not compatible with the operation code.
S	Syntax error: statement is not properly formed.
V	Value error: operand encountered in expression is improperly formed.

Several error messages are printed that are due to terminal error conditions:

NO SOURCE FILE PRESENT	The file specified in the ASM command does not exist on disk.
NO DIRECTORY SPACE	The disk directory is full; erase files that are not needed and retry.
SOURCE FILE NAME ERROR	Improperly formed ASM file name (e.g., it is specified with ? fields).
SOURCE FILE READ ERROR	Source file cannot be read properly by the assembler; execute a TYPE to determine the point of error.
OUTPUT FILE WRITE ERROR	Output files cannot be written properly; most likely cause is a full disk; erase and retry.
CANNOT CLOSE FILE	Output file cannot be closed; check to see if disk is write protected.

3.7 A Sample Session

The following session shows interaction with the assembler and debugger in the development of a simple assembly language program. The ↵ arrow represents a carriage return keystroke.

```
A>ASM SORT↵       Assemble SORT.ASM

CP/M ASSEMBLER - VER 1.0

015C      Next free address
003H USE FACTOR      Percent of table used 00 to ff (hexadecimal)
END OF ASSEMBLY

A>DIR SORT.*↵

SORT  ASM      Source file
SORT  BAK      Backup from last edit
SORT  PRN      Print file (contains tab characters)
SORT  HEX      Machine code file
A>TYPE SORT.PRN↵
                        Source line
                ;       SORT PROGRAM IN CP/M ASSEMBLY LANGUAGE
                ;       START AT THE BEGINNING OF THE TRANSIENT
                        PROGRAM AREA

Machine code location
  0100                  ORG             100H
```

Generated machine code

```
0100 214601SORT:    LXI  H,SW  ;ADDRESS SWITCH TOGGLE
0103 3601           MVI  M,1   ;SET TO 1 FOR FIRST ITERATION
0105 214701         LXI  H,I   ;ADDRESS INDEX
0108 3600           MVI  M,0   ;I = 0
                    ;
                    ;          COMPARE I WITH ARRAY SIZE
010A 7E      COMPL: MOV  A,M   ;A REGISTER = I
010B FE09           CPI N-1    ;CY SET IF I < (N-1)
010D D21901         JNC CONT ;CONTINUE IF I < = (N-2)
                    ;
                    ;          END OF ONE PASS THROUGH DATA
0110 214601         LXI  H,SW  ;CHECK FOR ZERO SWITCHES
0113 7EB7C20001     MOV A, M! ORA A! JNZ SORT ;END OF SORT IF SW=0
                    ;
0118 FF             RST 7      ;GO TO THE DEBUGGER INSTEAD OF REB
                    ;
                    ;          CONTINUE THIS PASS
Truncated           ;          ADDRESSING I, SO LOAD AV(I) INTO REGISTERS
0119
  5F16002148CONT:   MOV E, A! MVI D, 0! LXI H, AV! DAD D! DAD D
0121 4E792346       MOV C, M! MOV A, C! INX H! MOV B, M
                    ;          LOW ORDER BYTE IN A AND C, HIGH ORDER BYTE IN B
                    ;
                    ;          MOV H AND L TO ADDRESS AV(I+1)
0125 23             INX H
                    ;
                    ;          COMPARE VALUE WITH REGS CONTAINING AV (I)
0126 965778239E     SUB M! MOV D, A! MOV A, B! INX H! SBB M   ;SUBTRACT
                    ;
                    ;          BORROW SET IF AV(I+1) > AV(I)
012B DA3F01         JC   INCI   ;SKIP IF IN PROPER ORDER
                    ;
                    ;          CHECK FOR EQUAL VALUES
012E B2CA3F01       ORA D! JZ INCI ;SKIP IF AV(I) = AV(I+1)
0132 56702B5E       MOV D, M! MOV M, B! DCX H! MOV E, M
0136 712B722B73     MOV M, C! DCX H! MOV M, D! DCX H! MOV M, E
                    ;
                    ;          INCREMENT SWITCH COUNT
013B 21460134       LXI H,SW!  INR M
                    ;
                    ;          INCREMENT I
013F 21470134C3INCI: LXI H,I!  INR M! JMP COMP
                    ;
                    ;          DATA DEFINITION SECTION
0146 00      SW:    DB  0      ;RESERVE SPACE FOR SWITCH COUNT
0147         I:     DS  1      ;SPACE FOR INDEX
0148 050064001EAV:  DW  5, 100, 30, 50, 20, 7, 1000, 300, 100, -32767
000A =       N      EQU ($-AV)/2      ;COMPUTE N INSTEAD OF PRE
015C                END
A>TYPE SORT.HEX     Equate value

:10010000214601360121470136007EFE09D2190140
:100110002146017EB7C20001FF5F16002148011988
:10012000194E79234623965778239EDA3F01B2CAA7
```

Machine code in HEX format

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
:100130003F0156702B5E712B722B732146013421C7
:07014000470134C30A01006E
:10014800050064001E00320014000700E8032C01BB
:0401580064000180BE
:0000000000
A>DDT SORT.HEX↵      Start debug run

16K DDT VER 1.0
NEXT PC
015C 0000       Default address (no address on END statement)
-XP↵

P=0000 100↵     Change PC to 100

-UFFFF↵     Untrace for 65535 steps
                                                    Abort with rubout
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 LXI H,0146*0100
-T10↵     Trace 10₁₆ steps

C0Z0M0E0I0 A=01 B=0000 D=0000 H=0146 S=0100 P=0100 LXI H, 0146
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0146 S=0100 P=0103 MVI M, 01
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0146 S=0100 P=0105 LXI H, 0147
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0147 S=0100 P=0108 MVI M, 00
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0147 S=0100 P=010A MOV A, M
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=010B CPI 09
C1Z0M1E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=010D JNC 0119
C1Z0M1E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=0110 LXI H, 0146
C1Z0M1E0I0 A=00 B=0000 D=0000 H=0146 S=0100 P=0113 MOV A, M
C1Z0M1E0I0 A=01 B=0000 D=0000 H=0146 S=0100 P=0114 ORA A
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0146 S=0100 P=0115 JNZ 0100
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0146 S=0100 P=0100 LXI H, 0146
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0146 S=0100 P=0103 MVI M, 01
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0146 S=0100 P=0105 LXI H, 0147
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0147 S=0100 P=0108 MVI M, 00
C0Z0M0E0I0 A=01 B=0000 D=0000 H=0147 S=0100 P=010A MOV A, M*010B
-A10D                                                Stopped at 10BH

010D JC 119↵     Change to a jump on carry
0110↵

-XP↵

P=010B 100↵     Reset program counter back to beginning of program

-T10↵     Trace execution for 10H steps
                                                    Altered instruction
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=0100 LXI H,0146
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0146 S=0100 P=0103 MVI M,01
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0146 S=0100 P=0105 LXI H,0147
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=0108 MVI M,00
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=010A MOV A,M
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=010B CPI 09
C1Z0M1E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=010D JC 0119
C1Z0M1E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=0119 MOV E,A
C1Z0M1E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=011A MVI D,00
```

Machine code in HEX format

```
C1Z0M1E0I0 A=00 B=0000 D=0000 H=0147 S=0100 P=011C LXI H,0148
C1Z0M1E0I0 A=00 B=0000 D=0000 H=0148 S=0100 P=011F DAD D
C0Z0M1E0I0 A=00 B=0000 D=0000 H=0148 S=0100 P=0120 DAD D
C0Z0M1E0I0 A=00 B=0000 D=0000 H=0148 S=0100 P=0121 MOV C,M
C0Z0M1E0I0 A=00 B=0005 D=0000 H=0148 S=0100 P=0122 MOV A,C
C0Z0M1E0I0 A=05 B=0005 D=0000 H=0148 S=0100 P=1023 INX H
C0Z0M1E0I0 A=05 B=0005 D=0000 H=0149 S=0100 P=0124 MOV B,M*0125
-L100↵                                          Automatic breakpoint

 0100    LXI H,0146
 0103    MVI M,01
 0105    LXI H,0147
 0108    MVI M,00
 010A    MOV A,M
 010B    CPI 09
 010D    JC  0119
 0110    LXI H,0146
 0113    MOV A,M
 0114    ORA A
 0115    JNZ 0100
 -L↵
```

List some code from 100H

```
 0118    RST 07
 0119    MOV E,A
 011A    MVI D,00
 011C    LXI H,0148
```

List more

```
-Abort list with rubout
-G,11B↵     Start program from current PC (0125H) and run in real time to 11BH

*0127       Stopped with an external interrupt 7 from front panel (program was
                                                              looping indefinitely)
-T4↵   Look at looping program in trace mode
C0Z0M0E0I0 A=38 B=0064 D=0006 H=0156 S=0100 P=0127 MOV D,A
C0Z0M0E0I0 A=38 B=0064 D=3806 H=0156 S=0100 P=0128 MOV A,B
C0Z0M0E0I0 A=00 B=0064 D=3806 H=0156 S=0100 P=0129 INX H
C0Z0M0E0I0 A=00 B=0064 D=3806 H=0157 S=0100 P=012A SBB M*012B
-D148
                                  Data are sorted, but program does not stop.
0148 05 00 07 00 14 00 1E 00 ........
0150 32 00 64 00 64 00 2C 01 E8 03 01 80 00 00 00 00 2.D.D.,, ........

0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
-G0 ↵  Return to CP/M
A>DDT SORT.HEX↵     Reload the memory image

16K DDT VER. 1.0
NEXT  PC
015C 0000
-XP

P=0000 100↵     Set PC to beginning of program
```


ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
-L10D↵     List bad OPCODE

  010D    JNC 0119
  0110    LXI  H,0146
-Abort list with rubout
-A10D↵     Assemble new OPCODE

010D  JC 119↵

0110↵

-L100↵     List starting section of program

  0100    LXI  H,0146
  0103    MVI  M,01
  0105    LXI  H,0147
  0108    MVI  M,00
-Abort list with rubout
-A103↵     Change switch initialization to 00

0103  MVI M,0↵

0105↵

-^C     Return to CP/M with ctl-C (G0 works as well)

SAVE 1 SORT.COM↵     Save 1 page (256 bytes, from 100H to 1ffH) on disk in case
                     there is need to reload later
A>DDT SORT.COM↵      Restart DDT with saved memory image

16K DDT VER 1.0
NEXT  PC
0200 0100       COM file always starts with address 100H
-G↵     Run the program from PC=100H

*0118       Programmed stop (RST 7) encountered
-D148
                                        Data properly sorted
0148  05  00  07  00  14  00  1E  00 . . . . . .
0150  32  00  64  00  64  00  2C  01  E8  03  01  80  00  00  00  00  2.D.D. . . . . .

0160  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  . . . . . .
0170  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  . . . . . .

-G0 ↵ Return to CP/M
```

```
A>ED SORT.ASM↵      Make changes to original program

*N,0^Z0TT↵          Find next ",0"
        MVI             M, 0        ;I = 0

*-↵     Up one line in text
        LXI             H, I        ;ADDRESS INDEX

*-↵     Up another line
        MVI             M, 1        ;SET TO 1 FOR FIRST ITERATION

*KT↵    Kill line and type next line
        LXI             H, I        ;ADDRESS INDEX

*I↵     Insert new line
        MVI             M, 0        ;ZERO SW

*T↵
        LXI             H, I        ;ADDRESS INDEX

*NJNC^Z0T↵
        JNC*T↵
        CONT            ;CONTINUE IF I <= (N-2)

*-2DIC^Z0LT↵
        JC              CONT        ;CONTINUE IF I <= (N-2)

*E↵                     ——— Source from disk A
                        —— HEX to disk A
A>ASM SORT.AAZ↵         — Skip PRN file

CP/M ASSEMBLER - VER 1.0

015C        Next address to assemble
003H USE FACTOR
END OF ASSEMBLY

A>DDT SORT.HEX↵     Test program changes

16K DDT VER 1.0
NEXT PC
015C 0000
-G100↵

*0118
-D148↵
                                    Data sorted
0148  05  00  07  00  14  00  1E  00 ........
0150  32  00  64  00  64  00  2C  01  E8  03  01  80  00  00  00  00 2.D.D. . .
0160  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00 ........ .

-Abort with rubout

-G0↵    Return to CP/M—program checks OK.
```


ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

4

CP/M Dynamic Debugging Tool

4.1 Introduction

The DDT program allows dynamic interactive testing and debugging of programs generated in the CP/M environment. Invoke the debugger with a command of one of the following forms:

```
DDT
DDT filename.HEX
DDT filename.COM
```

where "filename" is the name of the program to be loaded and tested. In both cases, the DDT program is brought into main memory in place of the Console Command Processor (the user should refer to Chapter 5 for standard memory organization), and resides directly below the Basic Disk Operating System portion of CP/M. The BDOS starting address, located in the address field of the JMP instruction at location 5H, is altered to reflect the reduced Transient Program Area size.

The second and third forms of the DDT command perform the same actions as the first, except there is a subsequent automatic load of the specified HEX or COM file. The action is identical to the sequence of commands

```
DDT
Ifilename.HEX or Ifilename.COM
R
```

where the I and R commands set up and read the specified program to test. (The user should see the explanation of the I and R commands below for exact details.)

Upon initiation, DDT prints a sign-on message in the format

```
DDT VER m.m
```

where m.m is the revision number.

Following the sign-on message, DDT prompts the operator with the character "-" and waits for input commands from the console. The operator can type any of several single character commands, terminated by a carriage return to execute the command. Each line of input can be line-edited using the standard CP/M controls

rubout	remove the last character typed
ctl-U	remove the entire line, ready for retyping
ctl-C	system reboot.

Any command can be up to 32 characters in length (an automatic carriage return is inserted as the 33rd character), where the first character determines the command type

A	enter assembly language mnemonics with operands
D	display memory in hexadecimal and ASCII
F	fill memory with constant data
G	begin execution with optional breakpoints
I	set up a standard input file control block
L	list memory using assembler mnemonics
M	move a memory segment from source to destination
R	read program for subsequent testing
S	substitute memory values
T	trace program execution
U	untraced program monitoring
X	examine and optionally alter the CPU state.

The command character, in some cases, is followed by zero, one, two, or three hexadecimal values, which are separated by commas or single blank characters. All DDT numeric output is in hexadecimal form. The commands are not executed until the carriage return is typed at the end of the command.

At any point in the debug run, the operator can stop execution of DDT by using either a ctl-C or G0 (jmp to location 0000H), and save the current memory image by using a SAVE command of the form

```
SAVE n filename.COM
```

where n is the number of pages (256 byte blocks) to be saved on disk. The number of blocks is determined by taking the high order byte of the address in the TPA and converting this number to decimal. For example, if the highest address in the Transient Program Area is 1234H, the number of pages is 12H or 18 in decimal. The operator could type a ctl-C during the debug run, returning to the Console Command Processor level, followed by

```
SAVE 18 X.COM
```


ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

The memory image is saved as X.COM on the diskette and can be directly executed by typing the name X. If further testing is required, the memory image can be recalled by typing

```
DDT X.COM
```

which reloads the previously saved program from location 100H through page 18 (23FFH). The CPU state is not a part of the COM file; thus, the program must be restarted from the beginning to test it properly.

4.2 DDT Commands

The individual commands are detailed below. In each case, the operator must wait for the prompt character (-) before entering the command. If control is passed to a program under test and the program has not reached a breakpoint, control can be returned to DDT by executing a RST 7 from the front panel. In the explanation of each command, the command letter is shown in some cases with numbers separated by commas, and the numbers are represented by lower case letters. These numbers are always assumed to be in a hexadecimal radix and from one to four digits in length (longer numbers will be automatically truncated on the right).

Many of the commands operate upon a "CPU state" that corresponds to the program under test. The CPU state holds the registers of the program being debugged and initially contains zeroes for all registers and flags except for the program counter (P) and stack pointer (S), which default to 100H. The program counter is subsequently set to the starting address given in the last record of a HEX file if a file of this form is loaded (see the I and R commands).

4.2.1 The A (Assembly) Command

DDT allows in-line assembly language to be inserted into the current memory image using the A command, that takes the form

```
As
```

where s is the hexadecimal starting address for the inline assembly. DDT prompts the console with the address of the next instruction to fill and reads the console, looking for assembly language mnemonics (see the Intel 8080 Assembly Language Reference Card for a list of mnemonics), followed by register references and operands in absolute hexadecimal form. Each successive load address is printed before reading the console. The A command terminates when the first empty line is input from the console.

Upon completion of assembly language input, the operator can review the memory segment using the DDT disassembler (see the L command).

The user should note that the assembler/disassembler portion of DDT can be overlaid by the transient program being tested, in which case the DDT program responds with an error condition when the A and L commands are used.

4.2.2 The D (Display) Command

The D command allows the operator to view the contents of memory in hexadecimal and ASCII formats. The forms are

```
D
Ds
Ds,f
```

In the first case, memory is displayed from the current display address (initially 100H) and continues for 16 display lines. Each display line takes the form shown below

```
aaaa bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb cccccccccccccccc
```

where aaaa is the display address in hexadecimal and bb represents data present in memory starting at aaaa. The ASCII characters starting at aaaa are to the right (represented by the sequence of c's), where nongraphic characters are printed as a period (.). The user should note that both upper and lower case alphabetics are displayed, and will appear as upper case symbols on a console device that supports only upper case. Each display line gives the values of 16 bytes of data, with the first line truncated so that the next line begins at an address that is a multiple of 16.

The second form of the D command is similar to the first, except that the display address is first set to address s. The third form causes the display to continue from address s through address f. In all cases, the display address is set to the first address not displayed in this command, so that a continuing display can be accomplished by issuing successive D commands with no explicit addresses.

Excessively long displays can be aborted by pushing the return key.

4.2.3 The F (Fill) Command

The F command takes the form

```
Fs,f,c
```

where s is the starting address, f is the final address, and c is a hexadecimal byte constant. DDT stores the constant c at address s, increments the value of s and tests against f. If s exceeds f, the operation terminates, otherwise the operation is repeated. Thus, the fill command can be used to set a memory block to a specific constant value.

4.2.4 The G (Go) Command

A program is executed using the G command, with up to two optional breakpoint addresses. The G command takes the forms

```
G
Gs
Gs,b
Gs,b,c
G,b
G,b,c
```

The first form executes the program at the current value of the program counter in the current machine state, with no breakpoints set (the only way to regain control in DDT is through a RST 7 execution). The current program counter can be viewed by typing an X or XP command. The second form is similar to the first except that the program counter in the current machine state is set to address s before execution begins. The third form is the same as the second, except that program execution stops when address b is encountered (b must be in the area of the program under test). The instruction at location b is not executed when the breakpoint is encountered. The fourth form is identical to the third, except that two breakpoints are specified, one at b and the other at c. Encountering either breakpoint causes execution to stop, and both breakpoints are cleared. The last two forms take the program counter from the current machine state and set one and two breakpoints, respectively.

Execution continues from the starting address in real-time to the next breakpoint. There is no intervention between the starting address and the break address by DDT. If the program under test does not reach a breakpoint, control cannot return to DDT without executing a RST 7 instruction. Upon encountering a breakpoint, DDT stops execution and types

```
*d
```

where d is the stop address. The machine state can be examined at this point using the X (Examine) command. The operator must specify breakpoints that differ from the program counter address at the beginning of the G command. Thus, if the current program counter is 1234H, then the commands

```
G,1234
```

and

```
G400,400
```

both produce an immediate breakpoint without executing any instructions.

4.2.5 The I (Input) Command

The I command allows the operator to insert a file name into the default file control block at 5CH (the file control block created by CP/M for transient programs is placed at this location; see Chapter 5). The default FCB can be used by the program under test as if it had been passed by the CP/M Console Processor. The user should note that this file name is also used by DDT for reading additional HEX and COM files. The form of the I command is

```
Ifilename
```

or

```
Ifilename.typ
```

If the second form is used and the filetype is either HEX or COM, subsequent R commands can be used to read the pure binary or hex format machine code. (Section 4.2.8 gives further details.)

4.2.6 The L (List) Command

The L command is used to list assembly language mnemonics in a particular program region. The forms are

```
L
Ls
Ls,f
```

The first form lists twelve lines of disassembled machine code from the current list address. The second form sets the list address to s and then lists twelve lines of code. The last form lists disassembled code from s through address f. In all three cases, the list address is set to the next unlisted location in preparation for a subsequent L command. Upon encountering an execution breakpoint, the list address is set to the current value of the program counter (G and T commands). Again, long typeouts can be aborted using the return key during the list process.

4.2.7 The M (Move) Command

The M command allows block movement of program or data areas from one location to another in memory. The form is

```
Ms,f,d
```

where s is the start address of the move, f is the final address, and d is the destination address. Data are first removed from s to d, and both addresses are incremented. If s exceeds f, the move operation stops; otherwise, the move operation is repeated.

4.2.8 The R (Read) Command

The R command is used in conjunction with the I command to read COM and HEX files from the diskette into the transient program area in preparation for the debug run. The forms are

```
R
Rb
```

where b is an optional bias address that is added to each program or data address as it is loaded. The load operation must not overwrite any of the system parameters from 000H through 0FFH (i.e., the first page of memory). If b is omitted, then b=0000 is assumed. The R command requires a previous I command, specifying the name of a HEX or COM file. The load address for each record is obtained from each individual HEX record, while an assumed load address of 100H is used for COM files. The user should note that any number of R commands can be issued following the I command to reread the program under test, assuming the tested program does not destroy the default area at 5CH. Any file specified with the filetype "COM" is assumed to contain machine code in pure binary form (created with the LOAD or SAVE command), and all others are assumed to contain machine code in Intel hex format (produced, for example, with the ASM command.)

Recall that the command

```
DDT filename.filetype
```

which initiates the DDT program, is equivalent to the commands

```
DDT
-Ifilename.filetype
-R
```

Whenever the R command is issued, DDT responds with either the error indicator "?" (file cannot be opened, or a checksum error occurred in a HEX file), or with a load message taking the form

```
NEXT PC
nnnn pppp
```

where nnnn is the next address following the loaded program and pppp is the assumed program counter (100H for COM files, or taken from the last record if a HEX file is specified).

4.2.9 The S (Set) Command

The S command allows memory locations to be examined and optionally altered. The form of the command is

```
Ss
```

where s is the hexadecimal starting address for examination and alteration of memory. DDT responds with a numeric prompt, giving the memory location, along with the data currently held in memory. If the operator types a carriage return, the data are not altered. If a byte value is typed, the value is stored at the prompted address. In either case, DDT continues to prompt with successive addresses and values until either a period (.) is typed by the operator or an invalid input value is detected.

4.2.10 The T (Trace) Command

The T command allows selective tracing of program execution for 1 to 65535 program steps. The forms are

```
T
Tn
```

In the first case, the CPU state is displayed and the next program step is executed. The program terminates immediately, with the termination address displayed as

```
*hhhh
```

where hhhh is the next address to execute. The display address (used in the D command) is set to the value of H and L, and the list address (used in the L command) is set to hhhh. The CPU state at program termination can then be examined using the X command.

The second form of the T command is similar to the first, except that execution is traced for n steps (n is a hexadecimal value) before a program breakpoint occurs. A breakpoint can be forced in the trace mode by typing a rubout character. The CPU state is displayed before each program step is taken in trace mode. The format of the display is the same as described in the X command.

The user should note that program tracing is discontinued at the CP/M interface and resumes after return from CP/M to the program under test. Thus, CP/M functions that access I/O devices, such as the diskette drive, run in real-time, avoiding I/O timing problems. Programs running in trace mode execute approximately 500 times slower than real-time since DDT gets control after each user instruction is executed. Interrupt processing routines can be traced, but commands that use the breakpoint facility (G, T, and U) accomplish the break using an RST 7 instruction, which means that the tested program cannot use this interrupt location. Further, the trace mode always runs the tested program with interrupts enabled, which may cause problems if asynchronous interrupts are received during tracing.

The operator should use the return key to get control back to DDT during trace, rather than executing an RST 7, to ensure that the trace for current instruction is completed before interruption.

4.2.11 The U (Untrace) Command

The U command is identical to the T command except that intermediate program steps are not displayed. The untrace mode allows from 1 to 65535 (0FFFFH) steps to be executed in monitored mode and is used principally to retain control of an executing program while it reaches steady state conditions. All conditions of the T command apply to the U command.

4.2.12 The X (Examine) Command

The X command allows selective display and alteration of the current CPU state for the program under test. The forms are

```
X
Xr
```

where r is one of the 8080 CPU registers

C	Carry flag	(0/1)
Z	Zero flag	(0/1)
M	Minus flag	(0/1)
E	Even parity flag	(0/1)
I	Interdigit carry	(0/1)
A	Accumulator	(0-FF)
B	BC register pair	(0-FFFF)
D	DE register pair	(0-FFFF)
H	HL register pair	(0-FFFF)
S	Stack pointer	(0-FFFF)
P	Program counter	(0-FFFF)

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

In the first case, the CPU register state is displayed in the format

```
CfZfMfEfIf A=bb B=dddd D=dddd H=dddd S=dddd P=dddd inst
```

where f is a 0 or 1 flag value, bb is a byte value, and dddd is a double-byte quantity corresponding to the register pair. The "inst" field contains the disassembled instruction, which occurs at the location addressed by the CPU state's program counter.

The second form allows display and optional alteration of register values, where r is one of the registers given above (C, Z, M, E, I, A, B, D, H, S, or P). In each case, the flag or register value is first displayed at the console. The DDT program then accepts input from the console. If a carriage return is typed, the flag or register value is not altered. If a value in the proper range is typed, the flag or register value is altered. The user should note that BC, DE, and HL are displayed as register pairs. Thus, the operator types the entire register pair when B, C, or the BC pair is altered.

4.3 Implementation Notes

The organization of DDT allows certain nonessential portions to be overlaid to gain a larger transient program area for debugging large programs. The DDT program consists of two parts: the DDT nucleus and the assembler/disassembler module. The DDT nucleus is loaded over the Console Command Processor, and, although loaded with the DDT nucleus, the assembler/disassembler is overlayable unless used to assemble or disassemble.

In particular, the BDOS address at location 6H (address field of the JMP instruction at location 5H) is modified by DDT to address the base location of the DDT nucleus, which, in turn, contains a JMP instruction to the BDOS. Thus, programs that use this address field to size memory see the logical end of memory at the base of the DDT nucleus rather than the base of the BDOS.

The assembler/disassembler module resides directly below the DDT nucleus in the transient program area. If the A, L, T, or X commands are used during the debugging process, the DDT program again alters the address field at 6H to include this module, further reducing the logical end of memory. If a program loads beyond the beginning of the assembler/disassembler module, the A and L commands are lost (their use produces a "?" in response) and the trace and display (T and X) commands list the "inst" field of the display in hexadecimal, rather than as a decoded instruction.

4.4 An Example

The following example shows an edit, assemble, and debug for a simple program that reads a set of data values and determines the largest value in the set. The largest value is taken from the vector and stored into "LARGE" at the termination of the program

```
A>ED SCAN.ASM                Create source program;
                             "/" represents carriage return.
*I/
                ORG     1-00H       ;START  OF  TRANSIENT
                                    ;AREA/
                MVI     B, LEN      ;LENGTH OF VECTOR TO SCAN/
                MVI     C, 0        ;LARGER_RST VALUE SO FAR/
LOOP            LXI     H, VECT     ;BASE OF VECTOR/
LOOP:           MOV     A, M        ;GET VALUE/
                SUB     C           ;LARGER VALUE IN C?/
                JNC     NFOUND      ;JUMP IF LARGER VALUE NOT
                                    ;FOUND/
;               NEW LARGEST VALUE, STORE IT TO C/
                MOV     C, A
NFOUND          INX     H           ;TO NEXT ELEMENT/
                DCR     B           ;MORE TO SCAN?/
                JNZ     LOOP        ;FOR ANOTHER/
;
;               END OF SCAN, STORE C/
                MOV     A, C        ;GET LARGEST VALUE /
                STA     LARGE/
                JMP     0           ;REBOOT/
;/
:/              TEST DATA
VECT:           DB      2,0,4,3,5,6,1,5
LEN             EQU     $-VECT      ;LENGTH
LARGE:          DS      1           ;LARGEST VALUE ON EXIT/
                END/

!-Z
*B0P/
                ORG     100H        ;START OF TRANSIENT AREA
                MVI     B,LEN       ;LENGTH OF VECTOR TO SCAN
                MVI     C,0         ;LARGEST VALUE SO FAR
                LXI     H,VECT      ;BASE OF VECTOR
LOOP:           MOV     A,M         ;GET VALUE
                SUB     C           ;LARGER VALUE IN C?
                JNC     NFOUND      ;JUMP IF LARGER VALUE NOT
                                    ; FOUND
;               NEW LARGEST VALUE, STORE IT TO C
                MOV     C,A
NFOUND:         INX     H           ;TO NEXT ELEMENT
                DCR     B           ;MORE TO SCAN?
                JNZ     LOOP        ;FOR ANOTHER
;               END OF SCAN, STORE C
                MOV     A,C         ;GET LARGEST VALUE
                STA     LARGE
                JMP     0           ;REBOOT
;
;               TEST DATA
```


ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
VECT:           DB          2,0,4,3,5,6,1,5
LEN             EQU         $-VECT          ;LENGTH
LARGE:          DS          1               ;LARGEST VALUE ON EXIT
                END
*E↵  ← End of edit

A>ASM SCAN↵    Start Assembler

CP/M ASSEMBLER - VER 1.0

0122
002H USE FACTOR
END OF ASSEMBLY     Assembly complete; lock at program listing

A>TYPE SCAN.PRN ↵
 Code address    Source program
 0100 ←                          ORG  100H    ;START OF TRANSIENT AREA
 0100   0608                     MVI  B,LEN   ;LENGTH OF VECTOR TO SCAN
 0102   0E00  Machine code       MVI  C,0     ;LARGEST VALUE SO FAR
 0104   211901                   LXI  H,VECT. ;BASE OF VECTOR
 0107   7E            LOOP:     MOV  A,M     ;GET VALUE
 0108   91                       SUB  C       ;LARGER VALUE IN C?
 0109   D20D01                   JNC  NFOUND  ;JUMP IF LARGER VALUE NOT
                                              ;FOUND
                      ;          NEW LARGEST VALUE, STORE IT TO C
 010C   4F                       MOV  C, A

 010D   23            NFOUND:INX  H            ;TO NEXT ELEMENT
 010E   05                       DCR  B       ;MORE TO SCAN?
 010F   C20701                   JNZ  LOOP    ;FOR ANOTHER
                      ;
                      ;          END OF SCAN, STORE C
 0112   79                       MOV  A, C    ;GET LARGEST VALUE
 0113   322101                   STA  LARGE

 0116   C30000                   JMP  0       ;REBOOT
    Code—data listing ;
    truncated         ;          TEST DATA
 0119   0200040305    VECT:     DB   2,0,4,3,5,6,1,5
 0008   = Value of    LEN       EQU$-VECT    ;LENGTH
 0121     equate      LARGE:    DS   1       ;LARGEST VALUE ON EXIT
 0122                           END

A>DDT SCAN.HEX↵    Start debugger using hex format machine code

DDT VER 1.0
NEXT PC                                                  Next instruction
0121 0000                                                to execute at
-X↵   ← Last load address + 1                            PC=0

C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0000 OUT 7F
-XP↵                  ← Examine registers before debug run

P=0000 100↵    Change PC to 100

-X↵    Look at registers again
```

```
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI B,08
-L100
```

PC changed. Next instruction to execute at PC=100

```
 0100   MVI    B,08
 0102   MVI    C,00
 0104   LXI    H,0119
 0107   MOV    A,M
 0108   SUB    C
 0109   JNC    010D
 010C   MOV    C,A
 010D   INX    H
 010E   DCR    B
 010F   JNZ    0107
 0112   MOV    A,C
-L
```

Disassembled machine code at 100H (see source listing for comparison)

```
 0113   STA    0121
 0116   JMP    0000
 0119   STAX   B
 011A   NOP
 011B   INR    B
 011C   INX    B
 011D   DCR    B
 011E   MVI    B,01
 0120   DCR    B
 0121   LXI    D,2200
 0124   LXI    H,0200
```

A little more machine code. Note that program ends at location 116 with a JMP to 0000. Remainder of listing is assembly of data.

```
-A116
```
Enter in-line assembly mode to change the JMP to 0000 into a RST 7, which will cause the program under test to return to DDT if 116H is ever executed.

```
0116 RST 7

0117
```
(Single carriage return stops assemble mode)

```
-L113
```
List code at 113H to check that RST 7 was properly inserted

```
 0113   STA    0121
 0116   RST    07      in place of JMP
 0117   NOP
 0118   NOP
 0119   STAX   B
 011A   NOP
 011B   INR    B
 011C   INX    B
-
```

```
-X
```
Look at registers

```
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI B,08
-T
```
Execute Program for one stop. Initial CPU state, before / is executed

```
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI B,08*0102
-T
```
Automatic breakpoint

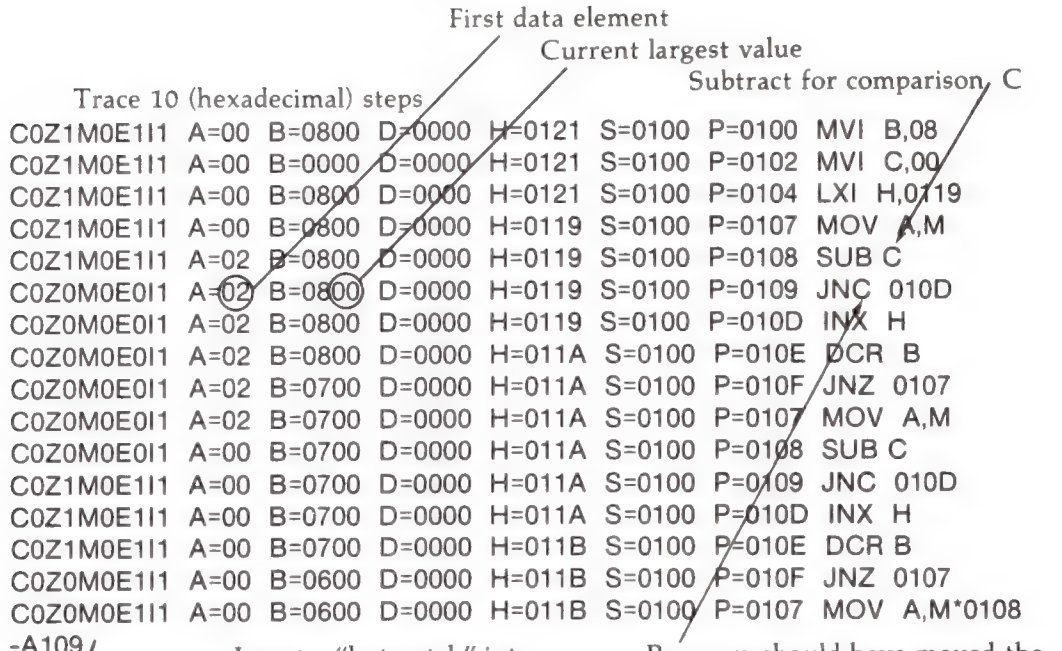
ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
      Trace one step again (note O8H in B)
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0000 S=0100 P=0102 MVI C,00*0104
-T↵
      Trace again (Register C is cleared)
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0000 S=0100 P=0104 LXI H,0119*0107
-T3↵ Trace three steps
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0119 S=0100 P=0107 MOV A,M
C0Z0M0E0I0 A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB C
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JNC 010D*010D
-D119↵ Display memory starting at 119H.      Automatic breakpoint at 10DH
0119 02 00 04 03 05 06 01 . Program data                Lowercase x
0120 05 11 00 22 21 00 02 7E EB 77 13 23 EB 0B 78 B1 . . . "! . . . W . # . . x
0130 C2 27 01 C3 03 29 00 00 00 00 00 00 00 00 00 00 . ' . . . ) . . . . . .
0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . . .
0150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . . .
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . . Data are displayed
0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . . inASCII with a "."
0180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . . in the position of
0190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . . nongraphic
01A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . . characters
01B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . . .
01C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . . .
-X↵
              Current CPU state
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=010D INX H
-T5↵
      Trace 5 steps from current CPU state
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=010D INX H
C0Z0M0E0I1 A=02 B=0800 D=0000 H=011A S=0100 P=010E DCR B
C0Z0M0E0I1 A=02 B=0700 D=0000 H=011A S=0100 P=010F JNZ 0107
C0Z0M0E0I1 A=02 B=0700 D=0000 H=011A S=0100 P=0107 MOV A,M
C0Z0M0E0I1 A=00 B=0700 D=0000 H=011A S=0100 P=0108 SUB C*0109
U5↵                                    Automatic breakpoint
      Trace without listing intermediate states
C0Z1M0E1I1 A=00 B=0700 D=0000 H=011A S=0100 P=0109 JNC 010D*0108
-X↵
                          CPU state at end of U5
C0Z0M0E1I1 A=04 B=0600 D=0000 H=011B S=0100 P=0108 SUB C
-G↵    Run program from current PC until completion (in real-time)

*0116      breakpoint at 116H, caused by executing RST 7 in machine code.
-X↵
      CPU state at end of program
C0Z1M0E1I1 A=00 B=0000 D=0000 H=0121 S=0100 P=0116 RST 07
-XP↵ Examine and change program counter
P=0116 100↵

-X↵

C0Z1M0E1I1 A=00 B=0000 D=0000 H=0121 S=0100 P=0100 MVI B,08
-T10↵
```


First data element
Current largest value
Subtract for comparison, C
Trace 10 (hexadecimal) steps
C0Z1M0E1I1 A=00 B=0800 D=0000 H=0121 S=0100 P=0100 MVI B,08
C0Z1M0E1I1 A=00 B=0000 D=0000 H=0121 S=0100 P=0102 MVI C,00
C0Z1M0E1I1 A=00 B=0800 D=0000 H=0121 S=0100 P=0104 LXI H,0119
C0Z1M0E1I1 A=00 B=0800 D=0000 H=0119 S=0100 P=0107 MOV A,M
C0Z1M0E1I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB C
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JNC 010D
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=010D INX H
C0Z0M0E0I1 A=02 B=0800 D=0000 H=011A S=0100 P=010E DCR B
C0Z0M0E0I1 A=02 B=0700 D=0000 H=011A S=0100 P=010F JNZ 0107
C0Z0M0E0I1 A=02 B=0700 D=0000 H=011A S=0100 P=0107 MOV A,M
C0Z0M0E0I1 A=00 B=0700 D=0000 H=011A S=0100 P=0108 SUB C
C0Z1M0E1I1 A=00 B=0700 D=0000 H=011A S=0100 P=0109 JNC 010D
C0Z1M0E1I1 A=00 B=0700 D=0000 H=011A S=0100 P=010D INX H
C0Z1M0E1I1 A=00 B=0700 D=0000 H=011B S=0100 P=010E DCR B
C0Z0M0E1I1 A=00 B=0600 D=0000 H=011B S=0100 P=010F JNZ 0107
C0Z0M0E1I1 A=00 B=0600 D=0000 H=011B S=0100 P=0107 MOV A,M*0108
-A109
Insert a "hot patch" into the machine code to change the JNC to JC
0109 JC 10D
010C
Program should have moved the value from A into C since A>C. Since this code was not executed, it appears that the JNC should have been a JC instruction
-G0
Stop DDT so that a version of the patched program can be saved
A>SAVE 1 SCAN.COM
Program resides on first page, so save 1 page.
A>DDT SCAN.COM
Restart DDT with the save memory image to continue testing
DDT VER 1.0
NEXT PC
0200 0100
-L100
List some code
0100 MVI B,08
0102 MVI C,00
0104 LXI H,0119
0107 MOV A,M
0108 SUB C
0109 JC 010D
Previous patch is present in X.COM
010C MOV C,A
010D INX H
010E DCR B
010F JNZ 0107
0112 MOV A,C
-XP
P=0100

```
-T10↵
     Trace to see how patched version operates          Data is moved from A to C
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI  B,08
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0000 S=0100 P=0102 MVI  C,00
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0000 S=0100 P=0104 LXI  H,0119
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0119 S=0100 P=0107 MOV  A,M
C0Z0M0E0I0 A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB  C
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JC   010D
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=010C MOV  C,A
C0Z0M0E0I1 A=02 B=0802 D=0000 H=0119 S=0100 P=010D INX  H
C0Z0M0E0I1 A=02 B=0802 D=0000 H=011A S=0100 P=010E DCR  B
C0Z0M0E0I1 A=02 B=0702 D=0000 H=011A S=0100 P=010F JNZ  0107
C0Z0M0E0I1 A=02 B=0702 D=0000 H=011A S=0100 P=0107 MOV  A,M
C0Z0M0E0I1 A=00 B=0702 D=0000 H=011A S=0100 P=0108 SUB  C
C1Z0M1E0I0 A=FE B=0702 D=0000 H=011A S=0100 P=0109 JC   010D
C1Z0M1E0I0 A=FE B=0702 D=0000 H=011A S=0100 P=010D INX  H
C1Z0M1E0I0 A=FE B=0702 D=0000 H=011B S=0100 P=010E DCR  B
C1Z0M0E1I1 A=FE B=0602 D=0000 H=011B S=0100 P=010F JNZ  0107*0107
-X↵                                          Breakpoint after 16 steps↵

C1Z0M0E1I1 A=FE B=0602 D=0000 H=011B S=0100 P=0107 MOV  A,M
-G,108↵    Run from current PC and breakpoint at 108H

*0108
-X↵
                  Next data item
C1Z0M0E1I1 A=04 B=0602 D=0000 H=011B S=0100 P=0108 SUB  C
-T↵
     Single step for a few cycles
C1Z0M0E1I1 A=04 B=0602 D=0000 H=011B S=0100 P=0108 SUB  C*0109
-T↵

C0Z0M0E0I1 A=02 B=0602 D=0000 H=011B S=0100 P=0109 JC 010D*010C
-X↵

C0Z0M0E0I1 A=02 B=0602 D=0000 H=011B S=0100 P=010C MOV  C,A
-G↵    Run to completion

*0116
-X↵

C0Z1M0E1I1 A=03 B=0003 D=0000 H=0121 S=0100 P=0116 RST  07
-S121↵    Look at the value of "LARGE"

 0121    03↵    Wrong value!

 0122    00↵

 0123    22↵

 0124    21↵
```

```
0125  00↵

0126  02↵

0127  7E↵ _        End of the S command

-L100↵

0100  MVI   B,08
0102  MVI   C,00
0104  LXI   H,0119
0107  MOV   A,M
0108  SUB   C
0109  JC    010D
010C  MOV   C,A
010D  INX   H
010E  DCR   B
010F  JNZ   0107
0112  MOV   A,C
-L↵                          } Review the code
0113  STA   0121
0116  RST   07
0117  NOP
0118  NOP
0119  STAX  B
011A  NOP
011B  INR   B
011C  INX   B
011D  DCR   B
011E  MVI   B,01
0120  DCR   B
-XP↵

P=0116 100↵    Reset the PC

-T↵
      Single step, and watch data values
C0Z1M0E1I1 A=03 B=0003 D=0000 H=0121 S=0100 P=0100 MVI B,08*0102
-T↵

C0Z1M0E1I1 A=03 B=0803 D=0000 H=0121 S=0100 P=0102 MVI C,00*0104
-T↵
          Count set ↘   ↙ "Largest" set
C0Z1M0E1I1 A=03 B=0800 D=0000 H=0121 S=0100 P=0104 LXI H,0119*0107
-T↵
                                    ↙ Base address of data set
C0Z1M0E1I1 A=03 B=0800 D=0000 H=0119 S=0100 P=0107 MOV A,M*0108
-T↵
                ↙ First data item brought to A
C0Z1M0E1I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB C*0109
-T↵

C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JC 010D*010C
-T↵
```


ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=010C MOV C,A*010D
-T↵
```

First data item moved to C correctly

```
C0Z0M0E0I1 A=02 B=0802 D=0000 H=0119 S=0100 P=010D INX H*010E
-T↵

C0Z0M0E0I1 A=02 B=0802 D=0000 H=011A S=0100 P=010E DCR B*010F
-T↵

C0Z0M0E0I1 A=02 B=0702 D=0000 H=011A S=0100 P=010F JNZ 0107*0107
-T↵

C0Z0M0E0I1 A=02 B=0702 D=0000 H=011A S=0100 P=0107 MOV A,M*0108
-T↵
```

Second data item brought to A

```
C0Z0M0E0I1 A=00 B=0702 D=0000 H=011A S=0100 P=0108 SUB C*0109
-T↵
```

Subtract destroys data value that was loaded!

```
C1Z0M1E0I0 A=FE B=0702 D=0000 H=011A S=0100 P=0109 JC 010D*010D
-T↵

C1Z0M1E0I0 A=FE B=0702 D=0000 H=011A S=0100 P=010D INX H*010E
-L100↵

 0100    MVI     B,08
 0102    MVI     C,00
 0104    LXI     H,0119
 0107    MOV     A,M
 0108    SUB     C
 0109    JC      010D
 010C    MOV     C,A
 010D    INX     H
 010E    DCR     B
 010F    JNZ     0107
 0112    MOV     A,C
 -A108↵
```

This should have been a CMP so that register A would not be destroyed.

```
0108 CMP C↵
```

Hot patch at 108H changes SUB to CMP

```
0109

-G0↵
```

Stop DDT for SAVE

```
A>SAVE 1 SCAN.COM↵
```

Save memory image

```
A>DDT SCAN.COM↵
```

Restart DDT

```
DDT VER 1.0
NEXT  PC
0200 0100
-XP↵

P=0100

-L116↵
```

```
0116    RST     07
0117    NOP
0118    NOP
0119    STAX    B
011A    NOP
-
```

Look at code to see if it was properly loaded (long typeout aborted with rubout)

```
-G,116↓     Run from 100H to completion

*0116
-XC↓     Look at carry (accidental typo)
C1↓
-X↓     Look at CPU state

C1Z1M0E1I1 A=06 B=0006 D=0000 H=0121 S=0100 P=0116 RST 07
-S121↓     Look at "large"—it appears to be correct.

0121 06↓

0122 00↓

0123 22

-G0↓     Stop DDT

A>ED SCAN.ASM↓     Re-edit the source program, and make both changes

*NSUB↓
*0LT↓
        ctl-Z↘      SUB     C           ;LARGER VALUE IN C?
*SSUB↑ZCMP↑Z0LT↓
                    CMP     C           ;LARGER VALUE IN C?
*
                    JNC     NFOUND      ;JUMP IF LARGER VALUE NOT FOUND
*SNC↑ZC↑Z0LT↓
                    JC      NFOUND      ;JUMP IF LARGER VALUE NOT FOUND
*E↓
                        Re-assemble, selecting source from disk A
A>ASM SCAN.AAZ↓ ←—Hex to disk A
                        Print to Z (selects no print file)
CP/M ASSEMBLER    VER 1.0

0122
002H USE FACTOR
END OF ASSEMBLY
```


ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
A>DDT SCAN.HEX↵     Re-run debugger to check changes

DDT VER 1.0
NEXT  PC
0121 0000
-L116↵

 0116    JMP     0000     Check to ensure end is still at 116H
 0119    STAX    B
 011A    NOP
 011B    INR     B
 - (rubout)

-G100,116↵     Go from beginning with breakpoint at end

*0116          Breakpoint reached
-D121↵         Look at "LARGE"
                                  Correct value computed
0121 06 00 22 21 00 02 7E EB 77 13 23 EB 0B 78 B1 . . '! . . . W . # . . X .
0130 C2 27 01 C3 03 29 00 00 00 00 00 00 00 00 00 00 .'. . . ) . . . . . .
0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 . . . . . .

- (rubout) Aborts long type-out

G0↵     Stop DDT, debug session complete.
```

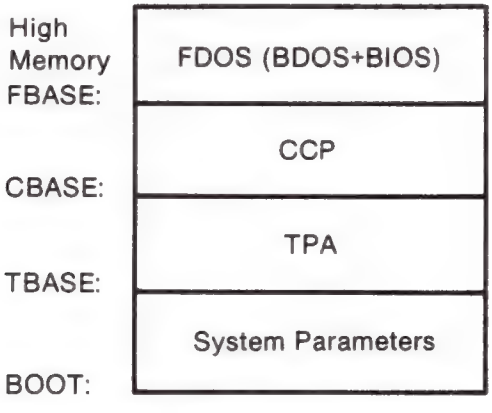
5

CP/M 2 System Interface

5.1 Introduction

This chapter describes CP/M, release 2, system organization including the structure of memory and system entry points. The intention is to provide necessary information required to write programs that operate under CP/M and that use the peripheral and disk I/O facilities of the system.

CP/M is logically divided into four parts, called the Basic I/O System (BIOS), the Basic Disk Operating System (BDOS), the Console Command Processor (CCP), and the Transient Program Area (TPA). The BIOS is a hardware-dependent module that defines the exact low level interface with a particular computer system that is necessary for peripheral device I/O. Although a standard BIOS is supplied by Digital Research, explicit instructions are provided for field reconfiguration of the BIOS to match nearly any hardware environment (see Chapter 6). The BIOS and BDOS are logically combined into a single module with a common entry point and referred to as the FDOS. The CCP is a distinct program that uses the FDOS to provide a human-oriented interface with the information that is cataloged on the backup storage device. The TPA is an area of memory (i.e., the portion that is not used by the FDOS and CCP) where various nonresident operating system commands and user programs are executed. The lower portion of memory is reserved for system information and is detailed in later sections. Memory organization of the CP/M system is shown below.



The exact memory addresses corresponding to BOOT, TBASE, CBASE, and FBASE vary from version to version and are described fully in Chapter 6. All standard CP/M versions, however, assume BOOT = 0000H, which is the base of random access memory. The machine code found at location BOOT performs a system "warm start," which loads and initializes the programs and variables necessary to return control to the CCP. Thus, transient programs need only jump to location BOOT to return control to CP/M at the command level. Further, the standard versions assume TBASE = BOOT+0100H, which is normally location 0100H. The principal entry point to the FDOS is at location BOOT+0005H (normally 0005H) where a jump to FBASE is found. The address field at BOOT+0006H (normally 0006H) contains the value of FBASE and can be used to determine the size of available memory, assuming that the CCP is being overlayed by a transient program.

Transient programs are loaded into the TPA and executed as follows. The operator communicates with the CCP by typing command lines following each prompt. Each command line takes one of the forms:

```
command
command file1
command file1 file2
```

where "command" is either a built-in function such as DIR or TYPE or the name of a transient command or program. If the command is a built-in function of CP/M, it is executed immediately. Otherwise, the CCP searches the currently addressed disk for a file by the name

```
command.COM
```

If the file is found, it is assumed to be a memory image of a program that executes in the TPA and thus implicitly originates at TBASE in memory. The CCP loads the COM file from the disk into memory starting at TBASE and can extend up to CBASE.

If the command is followed by one or two file specifications, the CCP prepares one or two file control block (FCB) names in the system parameter area. These optional FCBs are in the form necessary to access files through the FDOS and are described in the next section.

The transient program receives control from the CCP and begins execution, using the I/O facilities of the FDOS. The transient program is "called" from the CCP. Thus, it can simply return to the CCP upon completion of its processing or can jump to BOOT to pass control back to CP/M. In the first case, the transient program must not use memory above CBASE, while in the latter case, memory up through FBASE-1 can be used.

The transient program can use the CP/M I/O facilities to communicate with the operator's console and peripheral devices, including the disk subsystem. The I/O system is accessed by passing a function number and an information address to CP/M through the FDOS entry point at BOOT+0005H. In the case of a disk read, for example, the transient program sends the number corresponding to a disk read, along with the address of an FCB to the CP/M FDOS. The FDOS, in turn, performs the operation and returns with either a disk read completion indication or an error number indicating that the disk read was unsuccessful.

5.2 Operating System Call Conventions

This section provides detailed information for performing direct operating system calls from user programs. Many of the functions listed below, however, are accessed more simply through the I/O macro library provided with the MAC macro assembler and listed in the Digital Research manual entitled, *MAC Macro Assembler: Language Manual and Applications Guide.*

CP/M facilities that are available for access by transient programs fall into two general categories: simple device I/O and disk file I/O. The simple device operations include:

Read a Console Character

Write a Console Character

Read a Sequential Tape Character

Write a Sequential Tape Character

Write a List Device Character

Get or Set I/O Status

Print Console Buffer

Read Console Buffer

Interrogate Console Ready

The FDOS operations that perform disk I/O are

Disk System Reset

Drive Selection

File Creation

File Open

File Close

Directory Search

File Delete

File Rename

Random or Sequential Read

Random or Sequential Write

Interrogate Available Disks

Interrogate Selected Disk

Set DMA Address

Set/Reset File Indicators.

As mentioned above, access to the FDOS functions is accomplished by passing a function number and information address through the primary point at location BOOT+0005H. In general, the function number is passed in register C with the information address in the double byte pair DE. Single byte values are returned in register A, with double byte values returned in HL (a zero value is returned when the function number is out of range). For reasons of compatibility, register A = L and register B = H upon return in all cases. The user should note that the register passing conventions of CP/M agree with those of Intel's PL/M systems programming language. CP/M functions and their numbers are listed below.

0	System Reset
1	Console Input
2	Console Output
3	Reader Input
4	Punch Output
5	List Output
6	Direct Console I/O
7	Get I/O Byte
8	Set I/O Byte
9	Print String
10	Read Console Buffer
11	Get Console Status
12	Return Version Number
13	Reset Disk System
14	Select Disk
15	Open File
16	Close File
17	Search for First
18	Search for Next
19	Delete File
20	Read Sequential
21	Write Sequential
22	Make File
23	Rename File
24	Return Login Vector
25	Return Current Disk
26	Set DMA Address
27	Get Addr(Alloc)
28	Write Protect Disk
29	Get R/O Vector
30	Set File Attributes
31	Get Addr(Disk Parms)
32	Set/Get User Code
33	Read Random
34	Write Random
35	Compute File Size
36	Set Random Record
37	Reset Drive
40	Write Random with Zero Fill

(Functions 28 and 32 should be avoided in application programs to maintain upward compatibility with CP/M.)

Upon entry to a transient program, the CCP leaves the stack pointer set to an eight-level stack area with the CCP return address pushed onto the stack, leaving seven levels before overflow occurs. Although this stack is usually not used by a transient program (i.e., most transients return to the CCP through a jump to location 0000H), it is sufficiently large to make CP/M system calls since the FDOS switches to a local stack at system entry. The assembly language program segment below, for example, reads characters continuously until an asterisk is encountered, at which time control returns to the CCP (assuming a standard CP/M system with BOOT = 0000H).

```
BDOS    EQU     0005H       ;STANDARD CP/M ENTRY
CONIN   EQU     1           ;CONSOLE INPUT FUNCTION
;
        ORG     0100H       ;BASE OF TPA
NEXTC:  MVI     C,CONIN     ;READ NEXT CHARACTER
        CALL    BDOS        ;RETURN CHARACTER IN <A>
        CPI     '*'         ;END OF PROCESSING?
        JNZ     NEXTC       ;LOOP IF NOT
        RET                 ;RETURN TO CCP
        END
```


ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

CP/M implements a named file structure on each disk, providing a logical organization that allows any particular file to contain any number of records from completely empty to the full capacity of the drive. Each drive is logically distinct with a disk directory and file data area. The disk file names are in three parts: the drive select code, the filename consisting of one to eight nonblank characters, and the filetype consisting of zero to three nonblank characters. The filetype names the generic category of a particular file, while the filename distinguishes individual files in each category. The filetypes listed below name a few generic categories that have been established, although they are somewhat arbitrary.

ASM	Assembler Source	PLI	PL/I Source File
PRN	Printer Listing	REL	Relocatable Module
HEX	Hex Machine Code	TEX	TEX Formatter Source
BAS	Basic Source File	BAK	ED Source Backup
INT	Intermediate Code	SYM	SID Symbol File
COM	Command File	$$$	Temporary File

Source files are treated as a sequence of ASCII characters, where each "line" of the source file is followed by a carriage-return line-feed sequence (0DH followed by 0AH). Thus one 128-byte CP/M record could contain several lines of source text. The end of an ASCII file is denoted by a control-Z character (1AH) or a real end-of-file returned by the CP/M read operation. Control-Z characters embedded within machine code files (e.g., COM files) are ignored, however, and the end-of-file condition returned by CP/M is used to terminate read operations.

Files in CP/M can be thought of as a sequence of up to 65536 records of 128 bytes each, numbered from 0 through 65535, thus allowing a maximum of 8 megabytes per file. However, the user should note that although the records may be considered logically contiguous, they may not be physically contiguous in the disk data area. Internally, all files are divided into 16K byte segments called logical extents, so that counters are easily maintained as 8-bit values. The division into extents is discussed in the paragraphs that follow; however, they are not particularly significant for the programmer, since each extent is automatically accessed in both sequential and random access modes.

In the file operations starting with function number 15, DE usually addresses a file control block (FCB). Transient programs often use the default file control block area reserved by CP/M at location BOOT+005CH (normally 005CH) for simple file operations. The basic unit of file information is a 128-byte record used for all file operations; thus, a default location for disk I/O is provided by CP/M at location BOOT+0080H (normally 0080H), which is the initial default DMA address (see function 26). All directory operations take place in a reserved area that does not affect write buffers as was the case in release 1, with the exception of Search First and Search Next, where compatibility is required.

The FCB data area consists of a sequence of 33 bytes for sequential access and a series of 36 bytes in the case when the file is accessed randomly. The default FCB normally located at 005CH can be used for random access files, since the three bytes starting at BOOT+007DH are available for this purpose. The FCB format is shown with the following fields:

dr	f1	f2	/	/	f8	t1	t2	t3	ex	s1	s2	rc	d0	/	/	dn	cr	r0	r1	r2
00	01	02	...		08	09	10	11	12	13	14	15	16	...		31	32	33	34	35

where

dr	drive code (0-16) 0 => use default drive for file 1 => auto disk select drive A, 2 => auto disk select drive B, ... 16=> auto disk select drive P.
f1...f8	contain the file name in ASCII upper case, with high bit = 0
t1,t2,t3	contain the file type in ASCII upper case, with high bit = 0 t1', t2', and t3' denote the bit of these positions, t1' = 1 => Read/Only file, t2' = 1 => SYS file, no DIR list
ex	contains the current extent number, normally set to 00 by the user, but in range 0-31 during file I/O
s1	reserved for internal system use
s2	reserved for internal system use, set to zero on call to OPEN, MAKE, SEARCH
rc	record count for extent "ex," takes on values from 0-127
d0...dn	filled-in by CP/M, reserved for system use
cr	current record to read or write in a sequential file operation, normally set to zero by user
r0,r1,r2	optional random record number in the range 0-65535, with overflow to r2, r0, r1 constitute a 16-bit value with low byte r0, and high byte r1

Each file being accessed through CP/M must have a corresponding FCB, which provides the name and allocation information for all subsequent file operations. When accessing files, it is the programmer's responsibility to fill the lower 16 bytes of the FCB and initialize the cr field. Normally, bytes 1 through 11 are set to the ASCII character values for the file name and file type, while all other fields are zero.

FCBs are stored in a directory area of the disk, and are brought into central memory before the programmer proceeds with file operations (see the OPEN and MAKE functions). The memory copy of the FCB is updated as file operations take place and later recorded permanently on disk at the termination of the file operation (see the CLOSE command).

The CCP constructs the first 16 bytes of two optional FCBs for a transient by scanning the remainder of the line following the transient name, denoted by file1 and file2 in the prototype command line described above, with unspecified fields set to ASCII blanks. The first FCB is constructed at location BOOT+005CH and can be used as is for subsequent file operations. The second FCB occupies the d0 ... dn portion of the first FCB and must be moved to another area of memory before use. If, for example, the operator types

```
PROGNAME B:X.ZOT Y.ZAP
```


ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

the file PROGNAME.COM is loaded into the TPA and the default FCB at BOOT+005CH is initialized to drive code 2, file name X, and file type ZOT. The second drive code takes the default value 0, which is placed at BOOT+006CH, with the file name Y placed into location BOOT+006DH and file type ZAP located 8 bytes later at BOOT+0075H. All remaining fields through cr are set to zero. The user should note again that it is the programmer's responsibility to move this second file name and type to another area, usually a separate file control block, before opening the file that begins at BOOT+005CH, because the open operation will overwrite the second name and type.

If no file names are specified in the original command, the fields beginning at BOOT+005DH and BOOT+006DH contain blanks. In all cases, the CCP translates lower case alphabetics to upper case to be consistent with the CP/M file naming conventions.

As an added convenience, the default buffer area at location BOOT+0080H is initialized to the command line tail typed by the operator following the program name. The first position contains the number of characters, with the characters themselves following the character count. Given the above command line, the area beginning at BOOT+0080H is initialized as follows:

```
BOOT+0080H:

+00 +01 +02 +03 +04 +05 +06 +07 +08 +09 +A  +B  +C  +D  +E
E   ' '  'B' ':'  'X' '.'  'Z' 'O' 'T' ' '  'Y' '.'  'Z' 'A' 'P'
```

where the characters are translated to upper case ASCII with uninitialized memory following the last valid character. Again, it is the responsibility of the programmer to extract the information from this buffer before any file operations are performed, unless the default DMA address is explicitly changed.

Individual functions are described in detail in the pages that follow.

Function 0: System Reset

Entry Parameters:
Register C: 00H

The system reset function returns control to the CP/M operating system at the CCP level. The CCP reinitializes the disk subsystem by selecting and logging in disk drive A. This function has exactly the same effect as a jump to location BOOT.

Function 1: Console Input

Entry Parameters:
Register C: 01H

Returned Value:
Register A: ASCII Character

The console input function reads the next console character to register A. Graphic characters, along with carriage return, line feed, and back space (ctl-H) are echoed to the

console. Tab characters (ctl-I) move the cursor to the next tab stop. A check is made for start/stop scroll (ctl-S) and start/stop printer echo (ctl-P). The FDOS does not return to the calling program until a character has been typed, thus suspending execution if a character is not ready.

Function 2: Console Output

Entry Parameters:
Register C: 02H
Register E: ASCII Character

The ASCII character from register E is sent to the console device. As in function 1, tabs are expanded and checks are made for start/stop scroll and printer echo.

Function 3: Reader Input

Entry Parameters:
Register C: 03H

Returned Value:
Register A: ASCII Character

The Reader Input function reads the next character from the logical reader into register A (see the IOBYTE definition in Chapter 6). Control does not return until the character has been read.

Function 4: Punch Output

Entry Parameters:
Register C: 04H
Register E: ASCII Character

The Punch Output function sends the character from register E to the logical punch device.

Function 5: List Output

Entry Parameters:
Register C: 05H
Register E: ASCII Character

The List Output function sends the ASCII character in register E to the logical listing device.

Function 6: Direct Console I/O

Entry Parameters:
Register C: 06H
Register E: 0FFH (input) or char (output)

Returned Value:
Register A: char or status

Direct console I/O is supported under CP/M for those specialized applications where basic console input and output are required. Use of this function should, in general, be avoided since it bypasses all of CP/M's normal control character functions (e.g., control-S and control-P). Programs that perform direct I/O through the BIOS under previous releases of CP/M, however, should be changed to use direct I/O under BDOS so that they can be fully supported under future releases of MP/M and CP/M.

Upon entry to function 6, register E either contains hexadecimal FF, denoting a console input request, or an ASCII character. If the input value is FF, function 6 returns A = 00 if no character is ready, otherwise A contains the next console input character.

If the input value in E is not FF, function 6 assumes that E contains a valid ASCII character that is sent to the console.

Function 6 must not be used in conjunction with other console I/O functions.

Function 7: Get I/O Byte

Entry Parameters:
Register C: 07H

Returned Value:
Register A: I/O Byte Value

The Get I/O Byte function returns the current value of IOBYTE in register A. See Chapter 6 for IOBYTE definition.

Function 8: Set I/O Byte

Entry Parameters:
Register C: 08H
Register E: I/O Byte Value

The Set I/O Byte function changes the IOBYTE value to that given in register E.

Function 9: Print String

Entry Parameters:
Register C: 09H
Registers DE: String Address

The Print String function sends the character string stored in memory at the location given by DE to the console device, until a $ is encountered in the string. Tabs are expanded as in function 2, and checks are made for start/stop scroll and printer echo.

Function 10: Read Console Buffer

Entry Parameters:
Register C: 0AH
Registers DE: Buffer Address

Returned Value:
Console Characters in Buffer

The Read Buffer function reads a line of edited console input into a buffer addressed by registers DE. Console input is terminated when either input buffer overflows or a carriage return or line feed is typed. The Read Buffer takes the form:

```
DE: +0  +1  +2  +3  +4  +5  +6  +7  +8  . . .+n
    |mx|nc |c1 |c2 |c3 |c4 |c5 |c6 |c7 | . . .|??|
```

where mx is the maximum number of characters that the buffer will hold (1 to 255) and nc is the number of characters read (set by FDOS upon return), followed by the characters read from the console. If nc < mx, then uninitialized positions follow the last character, denoted by ?? in the above figure. A number of control functions are recognized during line editing:

rub/del	removes and echoes the last character
ctl-C	reboots when at the beginning of line
ctl-E	causes physical end of line
ctl-H	backspaces one character position
ctl-J	(line feed) terminates input line
ctl-M	(return) terminates input line
ctl-R	retypes the current line after new line
ctl-U	removes current line
ctl-X	same as ctl-U.

The user should also note that certain functions that return the carriage to the leftmost position (e.g., ctl-X) do so only to the column position where the prompt ended (in earlier

releases, the carriage returned to the extreme left margin). This convention makes operator data input and line correction more legible.

Function 11: Get Console Status

Entry Parameters:
Register C: 0BH

Returned Value:
Register A: Console Status

The Console Status function checks to see if a character has been typed at the console. If a character is ready, the value 0FFH is returned in register A. Otherwise a 00H value is returned.

Function 12: Return Version Number

Entry Parameters:
Register C: 0CH

Returned Value:
Registers HL: Version Number

Function 12 provides information that allows version independent programming. A two-byte value is returned, with H = 00 designating the CP/M release (H = 01 for MP/M), and L = 00 for all releases previous to 2.0. CP/M 2.0 returns a hexadecimal 20 in register L, with subsequent version 2 releases in the hexadecimal range 21, 22, through 2F. Using function 12, for example, the user can write application programs that provide both sequential and random access functions.

Function 13: Reset Disk System

Entry Parameters:
Register C: 0DH

The Reset Disk Function is used to programmatically restore the file system to a reset state where all disks are set to read/write (see functions 28 and 29), only disk drive A is selected, and the default DMA address is reset to BOOT+0080H. This function can be used, for example, by an application program that requires a disk change without a system reboot.

Function 14: Select Disk

Entry Parameters:
Register C: 0EH
Register E: Selected Disk

The Select Disk function designates the disk drive named in register E as the default disk for subsequent file operations, with E = 0 for drive A, 1 for drive B, and so on through 15, corresponding to drive P in a full 16 drive system. The drive is placed in an on-line status, which activates its directory until the next cold start, warm start, or disk system reset operation. If the disk medium is changed while it is on-line, the drive automatically goes to a read/only status in a standard CP/M environment (see function 28). FCBs that specify drive code zero (dr = 00H) automatically reference the currently selected default drive. Drive code values between 1 and 16, however, ignore the selected default drive and directly reference drives A through P.

Function 15: Open File

Entry Parameters:
Register C: 0FH
Registers DE: FCB Address

Returned Value:
Register A: Directory Code

The Open File operation is used to activate a file that currently exists in the disk directory for the currently active user number. The FDOS scans the referenced disk directory for a match in positions 1 through 14 of the FCB referenced by DE (byte s1 is automatically zeroed), where an ASCII question mark (3FH) matches any directory character in any of these positions. Normally, no question marks are included, and bytes ex and s2 of the FCB are zero.

If a directory element is matched, the relevant directory information is copied into bytes d0 through dn of the FCB, thus allowing access to the files through subsequent read and write operations. The user should note that an existing file must not be accessed until a successful open operation is completed. Upon return, the open function returns a directory code with the value 0 through 3 if the open was successful or 0FFH (255 decimal) if the file cannot be found. If question marks occur in the FCB, the first matching FCB is activated. Note that the current record (cr) must be zeroed by the program if the file is to be accessed sequentially from the first record.

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

Function 16: Close File

Entry Parameters:
Register C: 10H
Registers DE: FCB Address

Returned Value:
Register A: Directory Code

The Close File function performs the inverse of the open file function. Given that the FCB addressed by DE has been previously activated through an open or make function (see functions 15 and 22), the close function permanently records the new FCB in the referenced disk directory. The FCB matching process for the close is identical to the open function. The directory code returned for a successful close operation is 0, 1, 2, or 3, while a 0FFH (255 decimal) is returned if the file name cannot be found in the directory. A file need not be closed if only read operations have taken place. If write operations have occurred, however, the close operation is necessary to record the new directory information permanently.

Function 17: Search for First

Entry Parameters
Register C: 11H
Registers DE: FCB Address

Returned Value:
Register A: Directory Code

Search First scans the directory for a match with the file given by the FCB addressed by DE. The value 255 (hexadecimal FF) is returned if the file is not found; otherwise, 0, 1, 2, or 3 is returned indicating the file is present. When the file is found, the current DMA address is filled with the record containing the directory entry, and the relative starting position is A * 32 (i.e., rotate the A register left 5 bits, or ADD A five times). Although not normally required for application programs, the directory information can be extracted from the buffer at this position.

An ASCII question mark (63 decimal, 3F hexadecimal) in any position from f1 through ex matches the corresponding field of any directory entry on the default or auto-selected disk drive. If the dr field contains an ASCII question mark, the auto disk select function is disabled and the default disk is searched, with the search function returning any matched entry, allocated or free, belonging to any user number. This latter function is not normally used by application programs, but it allows complete flexibility to scan all current directory values. If the dr field is not a question mark, the s2 byte is automatically zeroed.

Function 18: Search for Next

Entry Parameters:
Register C: 12H

Returned Value:
Register A: Directory Code

The Search Next function is similar to the Search First function, except that the directory scan continues from the last matched entry. Similar to function 17, function 18 returns the decimal value 255 in A when no more directory items match.

Function 19: Delete File

Entry Parameters:
Register C: 13H
Registers DE: FCB Address

Returned Value:
Register A: Directory Code

The Delete File function removes files that match the FCB addressed by DE. The filename and type may contain ambiguous references (i.e., question marks in various positions), but the drive select code cannot be ambiguous, as in the Search and Search Next functions.

Function 19 returns a decimal 255 if the referenced file or files cannot be found; otherwise, a value in the range 0 to 3 is returned.

Function 20: Read Sequential

Entry Parameters:
Register C: 14H
Registers DE: FCB Address

Returned Value:
Register A: Directory Code

Given that the FCB addressed by DE has been activated through an open or make function (numbers 15 and 22), the Read Sequential function reads the next 128-byte record from the file into memory at the current DMA address. The record is read from position cr of the extent, and the cr field is automatically incremented to the next record position. If the cr field overflows, the next logical extent is automatically opened and the cr field is reset to zero in preparation for the next read operation. The value 00H is returned in the A register if the read operation was successful, while a nonzero value is returned if no data exist at the next record position (e.g., end-of-file occurs).

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

Function 21: Write Sequential

Entry Parameters:
Register C: 15H
Registers DE: FCB Address

Returned Value:
Register A: Directory Code

Given that the FCB addressed by DE has been activated through an open or make function (numbers 15 and 22), the Write Sequential function writes the 128-byte data record at the current DMA address to the file named by the FCB. The record is placed at position cr of the file, and the cr field is automatically incremented to the next record position. If the cr field overflows, the next logical extent is automatically opened and the cr field is reset to zero in preparation for the next write operation. Write operations can take place into an existing file, in which case, newly written records overlay those that already exist in the file. Register A = 00H upon return from a successful write operation, while a nonzero value indicates an unsuccessful write caused by a full disk.

Function 22: Make File

Entry Parameters:
Register C: 16H
Registers DE: FCB Address

Returned Value:
Register A: Directory Code

The Make File operation is similar to the open file operation except that the FCB must name a file that does not exist in the currently referenced disk directory (i.e., the one named explicitly by a nonzero dr code or the default disk if dr is zero). The FDOS creates the file and initializes both the directory and main memory value to an empty file. The programmer must ensure that no duplicate file names occur, and a preceding delete operation is sufficient if there is any possibility of duplication. Upon return, register A = 0, 1, 2, or 3 if the operation was successful and 0FFH (255 decimal) if no more directory space is available. The make function has the side effect of activating the FCB and thus a subsequent open is not necessary.

Function 23: Rename File

Entry Parameters:
Register C: 17H
Registers DE: FCB Address

Returned Value:
Register A: Directory Code

The Rename function uses the FCB addressed by DE to change all occurrences of the file named in the first 16 bytes to the file named in the second 16 bytes. The drive code dr

at position 0 is used to select the drive, while the drive code for the new file name at position 16 of the FCB is assumed to be zero. Upon return, register A is set to a value between 0 and 3 if the rename was successful and 0FFH (255 decimal) if the first file name could not be found in the directory scan.

Function 24: Return Log-in Vector

Entry Parameters:
Register C: 18H

Returned Value:
Registers HL: Log-in Vector

The log-in vector value returned by CP/M is a 16-bit value in HL, where the least significant bit of L corresponds to the first drive A and the high order bit of H corresponds to the sixteenth drive, labeled P. A 0 bit indicates that the drive is not on-line, while a 1 bit marks a drive that is actively on-line as a result of an explicit disk drive selection or an implicit drive select caused by a file operation that specified a nonzero dr field. The user should note that compatibility is maintained with earlier releases, since registers A and L contain the same values upon return.

Function 25: Return Current Disk

Entry Parameters:
Register C: 19H

Returned Value:
Register A: Current Disk

Function 25 returns the currently selected default disk number in register A. The disk numbers range from 0 through 15 corresponding to drives A through P.

Function 26: Set DMA Address

Entry Parameters:
Register C: 1AH
Registers DE: DMA Address

DMA is an acronym for Direct Memory Address, which is often used in connection with disk controllers that directly access the memory of the mainframe computer to transfer data to and from the disk subsystem. Although many computer systems use non-DMA access (i.e., the data are transferred through programmed I/O operations), the DMA address has, in CP/M, come to mean the address at which the 128-byte data record resides before a disk write and after a disk read. Upon cold start, warm start, or disk

system reset, the DMA address is automatically set to BOOT+0080H. The Set DMA function, however, can be used to change this default value to address another area of memory where the data records reside. Thus, the DMA address becomes the value specified by DE until it is changed by a subsequent Set DMA function, cold start, warm start, or disk system reset.

Function 27: Get ADDR(Alloc)

Entry Parameters:
Register C: 1BH

Returned Value:
Registers HL: ALLOC Address

An allocation vector is maintained in main memory for each on-line disk drive. Various system programs use the information provided by the allocation vector to determine the amount of remaining storage (see the STAT program). Function 27 returns the base address of the allocation vector for the currently selected disk drive. However, the allocation information may be invalid if the selected disk has been marked read/only. Although this function is not normally used by application programs, additional details of the allocation vector are found in Chapter 6.

Function 28: Write Protect Disk

Entry Parameters:
Register C: 1CH

The disk write protect function provides temporary write protection for the currently selected disk. Any attempt to write to the disk before the next cold or warm start operation produces the message:

```
BDOS ERR on d: R/O
```

Function 29: Get Read/Only Vector

Entry Parameters:
Register C: 1DH

Returned Value:
Registers HL: R/O Vector Value

Function 29 returns a bit vector in register pair HL, which indicates drives that have the temporary read-only bit set. As in function 24, the least significant bit corresponds to drive A, while the most significant bit corresponds to drive P. The R/O bit is set either by an explicit call to function 28 or by the automatic software mechanisms within CP/M that detect changed disks.

Function 30: Set File Attributes

Entry Parameters:
Register C: 1EH
Registers DE: FCB Address

Returned Value:
Register A: Directory Code

The Set File Attributes function allows programmatic manipulation of permanent indicators attached to files. In particular, the R/O and System attributes (t1′ and t2′) can be set or reset. The DE pair addresses an unambiguous file name with the appropriate attributes set or reset. Function 30 searches for a match and changes the matched directory entry to contain the selected indicators. Indicators f1′ through f4′ are not currently used, but may be useful for applications programs, since they are not involved in the matching process during file open and close operations. Indicators f5′ through f8′ and t3′ are reserved for future system expansion.

Function 31: Get ADDR(Disk Parms)

Entry Parameters:
Register C: 1FH

Returned Value:
Registers HL: DPB Address

The address of the BIOS resident disk parameter block is returned in HL as a result of this function call. This address can be used for either of two purposes. First, the disk parameter values can be extracted for display and space computation purposes, or transient programs can dynamically change the values of current disk parameters when the disk environment changes, if required. Normally, application programs will not require this facility.

Function 32: Set/Get User Code

Entry Parameters:
Register C: 20H
Register E: 0FFH (get) or
User Code (set)

Returned Value:
Register A: Current Code or
(no value)

An application program can change or interrogate the currently active user number by calling function 32. If register E = 0FFH, the value of the current user number is

returned in register A, where the value is in the range of 0 to 15. If register E is not 0FFH, the current user number is changed to the value of E (modulo 16).

Function 33: Read Random

Entry Parameters:
Register C: 21H
Registers DE: FCB Address

Returned Value:
Register A: Return Code

The Read Random function is similar to the sequential file read operation of previous releases, except that the read operation takes place at a particular record number, selected by the 24-bit value constructed from the 3-byte field following the FCB (byte positions r0 at 33, r1 at 34, and r2 at 35). The user should note that the sequence of 24 bits is stored with least significant byte first (r0), middle byte next (r1), and high byte last (r2). CP/M does not reference byte r2, except in computing the size of a file (function 35). Byte r2 must be zero, however, since a nonzero value indicates overflow past the end of file.

Thus, the r0, r1 byte pair is treated as a double-byte, or "word" value, which contains the record to read. This value ranges from 0 to 65535, providing access to any particular record of the 8-megabyte file. To process a file using random access, the base extent (extent 0) must first be opened. Although the base extent may or may not contain any allocated data, this ensures that the file is properly recorded in the directory and is visible in DIR requests. The selected record number is then stored in the random record field (r0, r1), and the BDOS is called to read the record. Upon return from the call, register A either contains an error code, as listed below, or the value 00, indicating the operation was successful. In the latter case, the current DMA address contains the randomly accessed record. The user should note that contrary to the sequential read operation, the record number is not advanced. Thus, subsequent random read operations continue to read the same record.

Upon each random read operation, the logical extent and current record values are automatically set. Thus, the file can be sequentially read or written, starting from the current randomly accessed position. However, the user should note that, in this case, the last randomly read record will be reread as one switches from random mode to sequential read and the last record will be rewritten as one switches to a sequential write operation. The user can, of course, simply advance the random record position following each random read or write to obtain the effect of a sequential I/O operation.

Error codes returned in register A following a random read are listed below.

01	reading unwritten data
02	(not returned in random mode)
03	cannot close current extent
04	seek to unwritten extent
05	(not returned in read mode)
06	seek past physical end of disk

Error codes 01 and 04 occur when a random read operation accesses a data block that has not been previously written or an extent that has not been created, which are equivalent conditions. Error code 03 does not normally occur under proper system

operation. If it does, it can be cleared by simply rereading or reopening extent zero as long as the disk is not physically wr e protected. Error code 06 occurs whenever byte r2 is nonzero under the current 2.0 release. Normally, nonzero return codes can be treated as missing data, with zero return codes indicating operation complete.

Function 34: Write Random

Entry Parameters:
Register C: 22H
Registers DE: FCB Address

Returned Value:
Register A: Return Code

The Write Random operation is initiated similarly to the Read Random call, except that data are written to the disk from the current DMA address. Further, if the disk extent or data block that is the target of the write has not yet been allocated, the allocation is performed before the write operation continues. As in the Read Random operation, the random record number is not changed as a result of the write. The logical extent number and current record positions of the file control block are set to correspond to the random record that is being written. Again, sequential read or write operations can begin following a random write, with the notation that the currently addressed record is either read or rewritten again as the sequential operation begins. The user can also simply advance the random record position following each write to get the effect of a sequential write operation. The user should note that, in particular, reading or writing the last record of an extent in random mode does not cause an automatic extent switch as it does in sequential mode.

The error codes returned by a random write are identical to the random read operation with the addition of error code 05, which indicates that a new extent cannot be created as a result of directory overflow.

Function 35: Compute File Size

Entry Parameters:
Register C: 23H
Registers DE: FCB Address

Returned Value:
Random Record Field Set

When computing the size of a file, the DE register pair addresses an FCB in random mode format (bytes r0, r1, and r2 are present). The FCB contains an unambiguous file name that is used in the directory scan. Upon return, the random record bytes contain the "virtual" file size, which is, in effect, the record address of the record following the end of the file. Following a call to function 35, if the high record byte r2 is 01, the file contains the maximum record count 65536. Otherwise, bytes r0 and r1 constitute a 16-bit value (r0 is the least significant byte, as before), which is the file size.

Data can be appended to the end of an existing file by simply calling function 35 to set the random record position to the end of file and then performing a sequence of random writes starting at the preset record address.

The virtual size of a file corresponds to the physical size when the file is written sequentially. If the file was created in random mode and "holes" exist in the allocation, the file may in fact contain fewer records than the size indicates. For example, if only the last record of an 8-megabyte file is written in random mode (i.e., record number 65535), the virtual size is 65536 records, although only one block of data is actually allocated.

Function 36: Set Random Record

Entry Parameters:
Register C: 24H
Registers DE: FCB Address

Returned Value:
Random Record Field Set

The Set Random Record function causes the BDOS automatically to produce the random record position from a file that has been read or written sequentially to a particular point. The function can be useful in two ways.

First, it is often necessary initially to read and scan a sequential file to extract the positions of various "key" fields. As each key is encountered, function 36 is called to compute the random record position for the data corresponding to this key. If the data unit size is 128 bytes, the resulting record position is placed into a table with the key for later retrieval. After scanning the entire file and tabulating the keys and their record numbers, the user can move instantly to a particular keyed record by performing a random read, using the corresponding random record number that was saved earlier. The scheme is easily generalized for variable record lengths, since the program need only store the buffer-relative byte position along with the key and record number to find the exact starting position of the keyed data at a later time.

A second use of function 36 occurs when switching from a sequential read or write over to random read or write. A file is sequentially accessed to a particular point in the file, function 36 is called, which sets the record number, and subsequent random read and write operations continue from the selected point in the file.

Function 37: Reset Drive

Entry Parameters:
Register C: 25H
Registers DE: Drive Vector

Returned Value:
Register A: 00H

The Reset Drive function allows resetting of specified drives. The passed parameter is a 16 bit vector of drives to be reset; the least significant bit is drive A:.

To maintain compatibility with MP/M, CP/M returns a zero value.

Function 40: Write Random With Zero Fill

Entry Parameters:
Register C: 28H
Registers DE: FCB Address

Returned Value:
Register A: Return Code

The Write Random With Zero Fill operation is similar to Function 34, with the exception that a previously unallocated block is filled with zeros before the data are written.

5.3 A Sample File-to-File Copy Program

The program shown below provides a relatively simple example of file operations. The program source file is created as COPY.ASM using the CP/M ED program and then assembled using ASM or MAC, resulting in a HEX file. The LOAD program is used to produce a COPY.COM file, which executes directly under the CCP. The program begins by setting the stack pointer to a local area and proceeds to move the second name from the default area at 006CH to a 33-byte file control block called DFCB. The DFCB is then prepared for file operations by clearing the current record field. At this point, the source and destination FCBs are ready for processing, since the SFCB at 005CH is properly set up by the CCP upon entry to the COPY program. That is, the first name is placed into the default FCB, with the proper fields zeroed, including the current record field at 007CH. The program continues by opening the source file, deleting any existing destination file, and creating the destination file. If all this is successful, the program loops at the label COPY until each record has been read from the source file and placed into the destination file. Upon completion of the data transfer, the destination file is closed and the program returns to the CCP command level by jumping to BOOT.

```
                        ;       sample file-to-file copy program
                        ;
                        ;       at the ccp level, the command
                        ;
                        ;            copy a:x.y b:u.v
                        ;
                        ;       copies the file named x.y from drive
                        ;       a to a file named u.v. on drive b.
                        ;
0000 =                  boot    equ  0000h     ; system reboot
0005 =                  bdos    equ  0005h     ; bdos entry point
005c =                  fcbl    equ  005ch     ; first file name
005c =                  sfcb    equ  fcbl      ; source fcb
006c =                  fcb2    equ  006ch     ; second file name
0080 =                  dbuff   equ  0080h     ; default buffer
0100 =                  tpa     equ  0100h     ; beginning of tpa
                        ;
0009 =                  printf  equ  9         ; print buffer func#
000f =                  openf   equ  15        ; open file func#
0010 =                  closef  equ  16        ; close file func#
```


ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
0013 =                  deletef equ 19          ; delete file func#
0014 =                  readf   equ 20          ; sequential read
0015 =                  writef  equ 21          ; sequential write
0016 =                  makef   equ 22          ; make file func#
                        ;
0100                            org tpa         ; beginning of tpa
0100 311b02                     lxi sp,stack    ; local stack
                        ;
                        ;       move second file name to dfcb
0103 0e10                       mvi c,16        ; half an fcb
0105 116c00                     lxi d,fcb2      ; source of move
0108 21da01                     lxi h,dfcb      ; destination fcb
010b 1a                 mfcb:   ldax d          ; source fcb
010c 13                         inx d           ; ready next
010d 77                         mov m,a         ; dest fcb
010e 23                         inx h           ; ready next
010f 0d                         dcr c           ; count 16...0
0110 c20b01                     jnz mfcb        ; loop 16 times
                        ;
                        ;       name has been removed, zero cr
0113 af                         xra a           ; a = 00h
0114 32fa01                     sta dfcbcr      ; current rec = 0
                        ;
                        ;       source and destination fcb's ready
                        ;
0117 115c00                     lxi d,sfcb      ; source file
011a cd6901                     call open       ; error if 255
011d 118701                     lxi d,nofile    ; ready message
0120 3c                         inr a           ; 255 becomes 0
0121 cc6101                     cz finis        ; done if no file
                        ;
                        ;       source file open, prep destination
0124 11da01                     lxi d,dfcb      ; destination
0127 cd7301                     call delete     ; remove if present
                        ;
012a 11da01                     lxi d,dfcb      ; destination
012d cd8201                     call make       ; create the file
0130 119601                     lxi d,nodir     ; ready message
0133 3c                         inr a           ; 255 becomes 0
0134 cc6101                     cz finis        ; done if no dir space
                        ;
                        ;       source file open, dest file open
                        ;       copy until end of file on source
                        ;
0137 115c00             copy:   lxi d,sfcb      ; source
013a cd7801                     call read       ; read next record
013d b7                         ora a           ; end of file?
013e c25101                     jnz eofile      ; skip write if so
                        ;
                        ;       not end of file, write the record
0141 11da01                     lxi d,dfcb      ; destination
0144 cd7d01                     call write      ; write record
0147 11a901                     lxi d,space     ; ready message
014a b7                         ora a           ; 00 if write ok
014b c46101                     cnz finis       ; end if so
```

```
014e c33701                     jmp  copy       ; loop until eof
                        ;
                        eofile: ; end of file, close destination
0151 11da01                     lxi  d,dfcb     ; destination
0154 cd6e01                     call close      ; 255 if error
0157 21bb01                     lxi  h,wrprot   ; ready message
015a 3c                         inr  a          ; 255 becomes 00
015b cc6101                     cz   finis      ; shouldn't happen
                        ;
                        ;       copy operation complete, end
015e 11cc01                     lxi  d,normal   ; ready message
                        ;
                        finis:  ; write message given by de, reboot
0161 0e09                       mvi  c,printf
0163 cd0500                     call bdos       ; write message
0166 c30000                     jmp  boot       ; reboot system
                        ;
                        ;       system interface subroutines
                        ;       (all return directly from bdos)
                        ;
0169 0e0f               open:   mvi  c,openf
016b c30500                     jmp  bdos
                        ;
016e 0e10               close:  mvi  c,closef
0170 c30500                     jmp  bdos
                        ;
0173 0e13               delete  mvi  c,deletef
0175 c30500                     jmp  bdos
                        ;
0178 0e14               read:   mvi  c,readf
017a c30500                     jmp  bdos
                        ;
017d 0e15               write:  mvi  c,writef
017f c30500                     jmp  bdos
                        ;
0182 0e16               make:   mvi  c,makef
0184 c30500                     jmp  bdos
                        ;
                        ;       console messages
0187 6e6f20f            nofile: db   'no source file$'
0196 6e6f209            nodir:  db   'no directory space$'
01a9 6f7574f            space:  db   'out of data space$'
01bb 7772695            wrprot: db   'write protected?$'
01cc 636f700            normal: db   'copy complete$'
                        ;
                        ;       data areas
01da                    dfcb:   ds   33         ; destination fcb
01fa =                  dfcbcr  equ  dfcb+32    ; current record
                        ;
01fb                            ds   32         ; 16 level stack
                        stack:
021b                            end
```


ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

The user should note that there are several simplifications in this particular program. First, there are no checks for invalid file names that could, for example, contain ambiguous references. This situation could be detected by scanning the 32-byte default area starting at location 005CH for ASCII question marks. A check should also be made to ensure that the file names have, in fact, been included (check locations 005DH and 006DH for nonblank ASCII characters). Finally, a check should be made to ensure that the source and destination file names are different. An improvement in speed could be obtained by buffering more data on each read operation. One could, for example, determine the size of memory by fetching FBASE from location 0006H and using the entire remaining portion of memory for a data buffer. In this case, the programmer simply resets the DMA address to the next successive 128-byte area before each read. Upon writing to the destination file, the DMA address is reset to the beginning of the buffer and incremented by 128 bytes to the end as each record is transferred to the destination file.

5.4 A Sample File Dump Utility

The file dump program shown below is slightly more complex than the simple copy program given in the previous section. The dump program reads an input file, specified in the CCP command line, and displays the content of each record in hexadecimal format at the console. Note that the dump program saves the CCP's stack upon entry, resets the stack to a local area, and restores the CCP's stack before returning directly to the CCP. Thus, the dump program does not perform and warm start at the end of processing.

```
                        ; DUMP program reads input file and displays hex
                        data
                        ;
0100                            org  100h
0005 =          bdos    equ  0005h =    ;bdos entry point
0001 =          cons    equ  1          ;read console
0002 =          typef   equ  2          ;type function
0009 =          printf  equ  9          ;buffer print entry
000b =          brkf    equ  11         ;break key function
                                        ;(true if char
000f =          openf   equ  15         ;file open
0014 =          readf   equ  20         ;read function
                ;
005c =          fcb     equ  5ch        ;file control block
                                        ;address
0080 =          buff    equ  80h        ;input disk buffer
                                        ;address
                ;
                ;       non graphic characters
000d =          cr      equ  0dh        ;carriage return
000a =          lf      equ  0ah        ;line feed
                ;
                ;       file control block definitions
005c =          fcbdn   equ  fcb+0      ;disk name
005d =          fcbfn   equ  fcb+1      ;file name
0065 =          fcbft   equ  fcb+9      ;disk file type (3
                                        ;characters)
0068 =          fcbrl   equ  fcb+12     ;file's current reel
                                        ;number
006b =          fcbrc   equ  fcb+15     ;file's record count (0 to
                                        ;128)128)
007c =          fcbcr   equ  fcb+32     ;current (next) record
                                        ;number (0
```

```
007d =                  fcbln   equ  fcb+33    ;fcb length
                        ;
                        ;       set up stack
0100 210000                     lxi  h,0
0103 39                         dad  sp
                        ;       entry stack pointer in hl from the ccp
0104 221502                     shld oldsp
                        ;       set sp to local stack area (restored at
                        ;       finis)
0107 315702                     lxi  sp,stktop
                        ;       read and print successive buffers
010a cdc101                     call setup      ;set up input file
010d feff                       cpi  255        ;255 if file not present
010f c21b01                     jnz  openok     ;skip if open is ok
                        ;
                        ;       file not there, give error message and
                        ;       return
0112 11f301                     lxi  d,opnmsg
0115 cd9c01                     call err
0118 c35101                     jmp  finis      ;to return
                        ;
                        openok: ;open operation ok, set buffer index to
                                ;end
011b 3e80                       mvi  a,80h
011d 321302                     sta  ibp        ;set buffer pointer to 80h
                        ;       hl contains next address to print
0120 210000                     lxi  h,0        ;start with 0000
                        ;
                        gloop:
0123 e5                         push h          ;save line position
0124 cda201                     call gnb
0127 e1                         pop  h          ;recall line position
0138 da5101                     jc   finis      ;carry set by gnb if end
                                                ;file
012b 47                         mov  b,a
                        ;       print hex values
                        ;       check for line fold
012c 7d                         mov  a,l
012d e60f                       ani  0fh        ;check low 4 bits
012f c24401                     jnz  nonum
                        ;       print line number
0132 cd7201                     call crlf
                        ;
                        ;       check for break key
0135 cd5901                     call break
                        ;       accum lsb = 1 if character ready
0138 0f                         rrc             ;into carry
0139 da5101                     jc   finis      ;don't print any more
                        ;
013c 7c                         mov  a,h
013d cd8f01                     call phex
0140 7d                         mov  a,l
0141 cd8f01                     call phex
                        nonum:
0144 23                         inx  h          ;to next line number
```


ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
0145 3e20                       mvi  a,' '
0147 cd6501                     call  pchar
014a 78                         mov a,b
014b cd8f01                     call  phex
014e c32301                     jmp  gloop
                        ;
                        finis:
                        ;       end of dump, return to cco
                        ;       (note that a jmp to 0000h reboots)
0151 cd7201                     call  crlf
0154 2a1502                     lhld  oldsp
0157 f9                         sphl
                        ;       stack pointer contains ccp's stack
                        ;       location
0158 c9                         ret             ;to the ccp
                        ;
                        ;
                        ;       subroutines
                        ;
                        break:  ;check break key (actually any key will
                                ;do)
0159 e5d5c5                     push h! push d! push b; environment
                                ; saved
015c 0e0b                       mvi  c,brkf
015e cd0500                     call  bdos
0161 c1d1e1                     pop b! pop d! pop h; environment
                                restored
0164 c9                         ret
                        ;
                        pchar:  ;print a character
0165 e5d5c5                     push h! push d! push b; saved
0168 0e02                       mvi  c,typef
016a 5f                         mov e,a
016b cd0500                     call  bdos
016e c1d1e1                     pop b! pop d! pop h; restored
0171 c9                         ret
                        ;
                        crlf:
0172 3e0d                       mvi  a,cr
0174 cd6501                     call  pchar
0177 3e0a                       mvi  a,lf
0179 cd6501                     call  pchar
017c c9                         ret
                        ;
                        ;
                        pnib:   ;print nibble in reg a
017d e60f                       ani  0fh        ;low 4 bits
017f fe0a                       cpi  10
0181 d28901                     jnc  p10
                        ;       less than or equal to 9
0184 c630                       adi  '0'
0186 c38b01                     jmp  prn
                        ;
                        ;       greater or equal to 10
0189 c637               p10:    adi  'a' - 10
```

```
018b cd6501             prn:    call  pchar
018e c9                         ret
                        ;
                        phex:   ;print hex char in reg a
018f f5                         pushpsw
0190 0f                         rrc
0191 0f                         rrc
0192 0f                         rrc
0193 0f                         rrc
0194 cd7d01                     call  pnib      ;print nibble
0197 f1                         pop   psw
0198 cd7d01                     call  pnib
019b c9                         ret
                        ;
                        err:    ;print error message
                        ;       d,e addresses message ending with "$"
019c 0e09                       mvi   c,printf          ;print buffer
                                                        ;function
019e cd0500                     call  bdos
01a1 c9                         ret
                        ;
                        ;
                        gnb:    ;get next byte
01a2 3a1302                     lda   ibp
01a5 fe80                       cpi   80h
01a7 c2b301                     jnz   g0
                        ;       read another buffer
                        ;
                        ;
01aa cdce01                     call  diskr
01ad b7                         ora   a         ;zero value if read ok
01ae cab301                     jz    g0        ;for another byte
                        ;       end of data, return with carry set for eof
01b1 37                         stc
01b2 c9                         ret
                        ;
                        g0:     ;read the byte at buff+reg a
01b3 5f                         mov e,a         ;ls byte of buffer index
01b4 1600                       mvi   d,0       ;double precision
                                                ; index to de
01b6 3c                         inr   a         ;index=index+1
01b7 321302                     sta   ibp       ;back to memory
                        ;       pointer is incremented
                        ;       save the current file address
01ba 218000                     lxi   h,buff
01bd 19                         dad   d
                        ;       absolute character address is in hl
01be 7e                         mov a,m
                        ;       byte is in the accumulator
01bf b7                         ora   a         ;reset carry bit
01c0 c9                         ret
                        ;
                        setup:  ;set up file
                        ;       open the file for input
01c1 af                         xra   a         ;zero to accum
```


ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
01c2 327c00                     sta   fcbcr     ;clear current record
                        ;
01c5 115c00                     lxi   d,fcb
01c8 0e0f                       mvi   c,openf
01ca cd0500                     call  bdos
                        ;       255 in accum if open error
01cd c9                         ret
                        ;
                        diskr:  ;read disk file record
01ce e5d5c5                     push h! push d! push b
01d1 115c00                     lxi   d,fcb
01d4 0e14                       mvi   c,readf
01d6 cd0500                     call  bdos
01d9 c1d1e1                     pop b! pop d! pop h
01dc c9                         ret
                        ;
                        ;       fixed message area
01dd 46494c0            signon: db    'file dump version 2.0$'
01f3 0d0a4e0            opnmsg: db    cr,lf,'no input file present on
                                      disk$'

                        ;       variable area
0213                    ibp:    ds    2         ;input buffer pointer
0215                    oldsp:  ds    2         ;entry sp value from ccp
                        ;
                        ;       stack area
0217                            ds    64        ;reserve 32 level stack
                        stktop:
                        ;
0257                            end
```

5.5 A Sample Random Access Program

This chapter concludes with an extensive example of random access operation. The program listed below performs the simple function of reading or writing random records upon command from the terminal. Given that the program has been created, assembled, and placed into a file labeled RANDOM.COM, the CCP level command

```
RANDOM X.DAT
```

starts the test program. The program looks for a file by the name X.DAT (in this particular case) and, if found, proceeds to prompt the console for input. If not found, the file is created before the prompt is given. Each prompt takes the form

```
next command?
```

and is followed by operator input, terminated by a carriage return. The input commands take the form

```
nW   nR   Q
```

where n is an integer value in the range 0 to 65535, and W, R, and Q are simple command characters corresponding to random write, random read, and quit processing, respectively. If the W command is issued, the RANDOM program issues the prompt

type data:.

The operator then responds by typing up to 127 characters, followed by a carriage return. RANDOM then writes the character string into the X.DAT file at record n. If the R command is issued, RANDOM reads record number n and displays the string value at the console. If the Q command is issued, the X.DAT file is closed, and the program returns to the CCP. In the interest of brevity, the only error message is

error, try again.

The program begins with an initialization section where the input file is opened or created, followed by a continuous loop at the label "ready" where the individual commands are interpreted. The default file control block at 005CH and the default buffer at 0080H are used in all disk operations. The utility subroutines then follow, which contain the principal input line processor, called "readc." This particular program shows the elements of random access processing, and can be used as the basis for further program development.

Sample Random Access Program for CP/M 2.0

```
0100                    org     100h    ;base of tpa
                        ;
0000 =          reboot  equ     0000h   ;system reboot
0005 =          bdos    equ     0005h   ;bdos entry point
                        ;
0001 =          coninp  equ     1       ;console input function
0002 =          conout  equ     2       ;console output function
0009 =          pstring equ     9       ;print string until '$'
000a =          rstring equ     10      ;read console buffer
000c =          version equ     12      ;return version number
000f =          openf   equ     15      ;file open function
0010 =          closef  equ     16      ;close function
0016 =          makef   equ     22      ;make file function
0021 =          readr   equ     33      ;read random
0022 =          writer  equ     34      ;write random
                        ;
005c =          fcb     equ     005ch   ;default file control
                                        ;block
007d =          ranrec  equ     fcb+33  ;random record position
007f =          ranovf  equ     fcb+35  ;high order (overflow)
                                        ;byte
0080 =          buff    equ     0080h   ;buffer address
                        ;
000d =          cr      equ     0dh     ;carriage return
000a =          lf      equ     0ah     ;line feed
                        ;
```

Load SP, Set-Up File for Random Access

```
0100 31bc00             lxi     sp,stack
                        ;
                        ;       version 2.0
0103 0e0c               mvi     c,version
```

```
0105 cd0500                     call    bdos
0108 fe20                       cpi     20h     ;version 2.0 or better?
010a d21600                     jnc     versok
                        ;       bad version, message and go back
010d 111b00                     lxi     d,badver
0110 cdda00                     call    print
0113 c30000                     jmp     reboot
                        ;
                        versok:
                        ;       correct version for random access
0116 0e0f                       mvi     c,openf ;open default fcb
0118 115c00                     lxi     d,fcb
011b cd0500                     call    bdos
011e 3c                         inr     a       ;err 255 becomes zero
011f c23700                     jnz     ready
                        ;
                        ;       cannot open file, so create it
0122 0e16                       mvi     c,makef
0124 115c00                     lxi     d,fcb
0127 cd0500                     call    bdos
012a 3c                         inr     a       ;err 255 becomes zero
012b c23700                     jnz     ready
                        ;
                        ;       cannot create file, directory full
012e 113a00                     lxi     d,nospace
0131 cdda00                     call    print
0134 c30000                     jmp     reboot  ;back to ccp
```

Loop Back to Ready After Each Command

```
                        ;
                        ready:
                        ;       file is ready for processing
                        ;
0137 cde500                     call    readcom ;read next command
013a 227d00                     shld    ranrec  ;store input record#
013d 217f00                     lxi     h,ranovf
0140 3600                       mvi     m,0     ;clear high byte if set
0142 fe51                       cpi     'Q'     ;quit?
0144 c25600                     jnz     notq
                        ;
                        ;       quit processing, close file
0147 0e10                       mvi     c,closef
0149 115c00                     lxi     d,fcb
014c cd0500                     call    bdos
014f 3c                         inr     a       ;err 255 becomes 0
0150 cab900                     jz      error   ;error message, retry
0153 c30000                     jmp     reboot  ;back to ccp
                        ;
```

End of Quit Command, Process Write

```
                        notq:
                        ;       not the quit command, random write?
0156 fe57                       cpi     'W'
0158 c28900                     jnz     notw
                        ;
                        ;       this is a random write, fill buffer until cr
015b 114d00                     lxi     d,datmsg
015e cdda00                     call    print   ;data prompt
0161 0e7f                       mvi     c,127   ;up to 127 characters
0163 218000                     lxi     h,buff  ;destination
                        rloop:  ;read next character to buff
0166 c5                         push    b       ;save counter
0167 e5                         push    h       ;next destination
0168 cdc200                     call    getchr  ;character to a
016b e1                         pop     h       ;restore counter
016c c1                         pop     b       ;restore next to fill
016d fe0d                       cpi     cr      ;end of line?
016f ca7800                     jz      erloop
                        ;       not end, store character
0172 77                         mov     m,a
0173 23                         inx     h       ;next to fill
0174 0d                         dcr     c       ;counter goes down
0175 c26600                     jnz     rloop   ;end of buffer?
                        erloop:
                        ;       end of read loop, store 00
0178 3600                       mvi     m,0
                        ;
                        ;       write the record to selected record number
017a 0e22                       mvi     c,writer
017c 115c00                     lxi     d,fcb
017f cd0500                     call    bdos
0182 b7                         ora     a       ;error code zero?
0183 c2b900                     jnz     error   ;message if not
0186 c33700                     jmp     ready   ;for another record
                        ;
```

End of Write Command, Process Read

```
                        notw:
                        ;       not a write command, read record?
0189 fe52                       cpi     'R'
018b c2b900                     jnz     error   ;skip if not
                        ;
                        ;       read random record
018e 0e21                       mvi     c,readr
0190 115c00                     lxi     d,fcb
0193 cd0500                     call    bdos
0196 b7                         ora     a       ;return code 00?
0197 c2b900                     jnz     error
                        ;
                        ;       read was successful, write to console
```


ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
019a cdcf00             call    crlf    ;new line
019d 0e80               mvi     c,128   ;max 128 characters
019f 218000             lxi     h,buff  ;next to get
                wloop:
01a2 7e                 mov     a,m     ;next character
01a3 23                 inx     h       ;next to get
01a4 e67f               ani     7fh     ;mask parity
01a6 ca3700             jz      ready   ;for another command
                                        ;if 00
01a9 c5                 push    b       ;save counter
01aa e5                 push    h       ;save next to get
01ab fe20               cpi     ' '     ;graphic?
01ad d4c800             cnc     putchr  ;skip output if not
01b0 e1                 pop     h
01b1 c1                 pop     b
01b2 0d                 dcr     c       ;count=count-1
01b3 c2a200             jnz     wloop
01b6 c33700             jmp     ready
```

End of Read Command, All Errors End Up Here

```
                ;
                error:
01b9 115900             lxi     d,errmsg
01bc cdda00             call    print
01bf c33700             jmp     ready
                ;
```

Utility Subroutines for Console I/O

```
                getchr:
                        ;read next console character to a
01c2 0e01               mvi     c,coninp
01c4 cd0500             call    bdos
01c7 c9                 ret
                ;
                putchr:
                        ;write character from a to console
01c8 0e02               mvi     c,conout
01ca 5f                 mov     e,a     ;character to send
01cb cd0500             call    bdos    ;send character
01ce c9                 ret
                ;
                crlf:
                        ;send carriage return line feed
01cf 3e0d               mvi     a,cr    ;carriage return
01d1 cdc800             call    putchr
01d4 3e0a               mvi     a,lf    ;line feed
01d6 cdc800             call    putchr
01d9 c9                 ret
```

```
                        ;
                        print:
                                ;print the buffer addressed by de until $
01da d5                         push    d
01db cdcf00                     call    crlf
01de d1                         pop     d       ;new line
01df 0e09                       mvi     c,pstring
01e1 cd0500                     call    bdos    ;print the string
01e4 c9                         ret
                        ;
                        readcom:
                                ;read the next command line to the conbuf
01e5 116b00                     lxi     d,prompt
01e8 cdda00                     call    print   ;command?
01eb 0e0a                       mvi     c,rstring
01ed 117a00                     lxi     d,conbuf
01f0 cd0500                     call    bdos    ;read command line
                        ;       command line is present, scan it
01f3 210000                     lxi     h,0     ;start with 0000
01f6 117c00                     lxi     d,conlin ;command line
01f9 1a                 readc:  ldax    d       ;next command
                                                ;character
01fa 13                         inx     d       ;to next command
                                                ;position
01fb b7                         ora     a       ;cannot be end of
                                                ;command
01fc c8                         rz
                        ;       not zero, numeric?
01fd d630                       sui     '0'
01ff fe0a                       cpi     10      ;carry if numeric
0201 d21300                     jnc     endrd
                        ;       add-in next digit
0204 29                         dad     h       ;*2
0205 4d                         mov     c,l
0206 44                         mov     b,h     ;bc = value * 2
0207 29                         dad     h       ;*4
0208 29                         dad     h       ;*8
0209 09                         dad     b       ;*2 + *8 = *10
020a 85                         add     l       ;+digit
020b 6f                         mov     l,a
020c d2f900                     jnc     readc   ;for another char
020f 24                         inr     h       ;overflow
0210 c3f900                     jmp     readc   ;for another char
                        endrd:
                        ;       end of read, restore value in a
0213 c630                       adi     '0'     ;command
0215 fe61                       cpi     'a'     ;translate case?
0217 d8                         rc
                        ;       lower case, mask lower case bits
0218 e65f                       ani     101$1111b
021a c9                         ret
                        ;
```


ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

String Data Area for Console Messages

```
                        badver:
021b 536f79                     db      'sorry, you need cp/m version 2$'
                        nospace:
023a 4e6f29                     db      'no directory space$'
                        datmsg:
024d 547970                     db      'type data: $'
                        errmsg:
0259 457272                     db      'error, try again.$'
                        prompt:
026b 4e6570                     db      'next command? $'
                        ;
```

Fixed and Variable Data Area

```
027a 21         conbuf: db      conlen  ;length of console buffer
027b            consiz: ds      1       ;resulting size after read
027c            conlin: ds      32      ;length 32 buffer
0021 =          conlen  equ     $-consiz
                ;
029c                    ds      32      ;16 level stack
                stack:
02bc                    end
```

Again, major improvements could be made to this particular program to enhance its operation. In fact, with some work, this program could evolve into a simple data base management system. One could, for example, assume a standard record size of 128 bytes, consisting of arbitrary fields within the record. A program, called GETKEY, could be developed that first reads a sequential file and extracts a specific field defined by the operator. For example, the command

```
GETKEY NAMES.DAT LASTNAME 10 20
```

would cause GETKEY to read the data base file NAMES.DAT and extract the "LASTNAME" field from each record, starting in position 10 and ending at character 20. GETKEY builds a table in memory consisting of each particular LASTNAME field, along with its 16-bit record number location within the file. The GETKEY program then sorts this list and writes a new file, called LASTNAME.KEY, which is an alphabetical list of LASTNAME fields with their corresponding record numbers. (This list is called an *inverted index* in information retrieval parlance.)

If the programmer were to rename the program shown above as QUERY and massage it so that it reads a sorted key file into memory, the command line might appear as

```
QUERY NAMES.DAT LASTNAME.KEY.
```

Instead of reading a number, the QUERY program reads an alphanumeric string that is a particular key to find in the NAMES.DAT data base. Since the LASTNAME.KEY list is sorted, one can find a particular entry rapidly by performing a "binary search," similar to looking up a name in the telephone book. That is, starting at both ends of the list, one examines the entry halfway in between and, if not matched, splits either the upper half or

the lower half for the next search. The user will quickly reach the item he or she is looking for and find the corresponding record number. The user should fetch and display this record at the console, just as was done in the program shown above.

With some more work, the user can allow a fixed grouping size that differs from the 128-byte record shown above. This is accomplished by keeping track of the record number as well as the byte offset within the record. Knowing the group size, one randomly accesses the record containing the proper group, offset to the beginning of the group within the record read sequentially until the group size has been exhausted.

Finally, one can improve QUERY considerably by allowing boolean expressions, which compute the set of records that satisfy several relationships, such as a LASTNAME between HARDY and LAUREL and an AGE lower than 45. Display all the records that fit this description. Finally, if the user's lists are getting too big to fit into memory, he or she should randomly access key files from the disk as well.

5.6 System Function Summary

FUNCTION NUMBER		FUNCTION NAME	INPUT	OUTPUT
Decimal	Hex			
0	0	System Reset	C = 00H	none
1	1	Console Input	C = 01H	A = ASCII char
2	2	Console Output	E = char	none
3	3	Reader Input		A = ASCII char
4	4	Punch Output	E = char	none
5	5	List Output	E = char	none
6	6	Direct Console I/O	C = 06H E = OFFH (input) or OFEH (status) or char (output)	A = char or status (no value)
7	7	Get I/O Byte	none	A = I/O Byte Value
8	8	Set I/O Byte	E = I/O Byte	none
9	9	Print String	DE = Buffer Address	none
10	A	Read Console Buffer	DE = Buffer	Console Characters in Buffer
11	B	Get Console Status	none	A = 00/non zero
12	C	Return Version Number	none	HL: Version Number
13	D	Reset Disk System	none	none
14	E	Select Disk	E = Disk Number	none
15	F	Open File	DE = FCB Address	FF if not found
16	10	Close File	DE = FCB Address	FF if not found

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

17	11	Search For First	DE = FCB Address	A = Directory Code
18	12	Search For Next	none	A = Directory Code
19	13	Delete File	DE = FCB Address	A = none
20	14	Read Sequential	DE = FCB Address	A = Error Code
21	15	Write Sequential	DE = FCB Address	A = Error Code
22	16	Make File	DE = FCB Address	A = FF if no DIR Space
23	17	Rename File	DE = FCB Address	A = FF if not found
24	18	Return Login Vector	none	HL = Login Vector*
25	19	Return Current Disk	none	A = Current Disk Number
26	1A	Set DMA Address	DE = DMA Address	none
27	1B	Get ADDR (ALLOC)	none	HL = ALLOC Address*
28	1C	Write Protect Disk	none	none
29	1D	Get Read/only Vector	none	HL = R/O Vector Value*
30	1E	Set File Attributes	DE = FCB Address	A = none
31	1F	Get ADDR (Disk Parms)	none	HL = DPB Address
32	20	Set/Get User Code	E = 0FFH for Get E = 00 to 0FH for Set	User Number
33	21	Read Random	DE = FCB Address	A = Error Code
34	22	Write Random	DE = FCB Address	A = Error Code
35	23	Compute File Size	DE = FCB Address	r0, r1, r2
36	24	Set Random Record	DE = FCB Address	r0, r1, r2
37	25	Reset Drive	DE = Drive Vector	A = 0
38	26	Access Drive	not supported	
39	27	Free Drive	not supported	
40	28	Write Random with Fill	DE = FCB	A = Error Code

*Note that A = L, and B = H upon return.

6

CP/M 2 Alteration

6.1 Introduction

The standard CP/M system assumes operation on an Intel MDS-800 microcomputer development system, but is designed so the user can alter a specific set of subroutines that define the hardware operating environment.

Although standard CP/M 2 is configured for single density floppy disks, field-alteration features allow adaptation to a wide variety of disk subsystems from single drive minidisks through high-capacity, "hard disk" systems. To simplify the following adaptation process, it is assumed that CP/M 2 will first be configured for single density floppy disks where minimal editing and debugging tools are available. If an earlier version of CP/M is available, the customizing process is eased considerably. In this latter case, the user may wish to review the system generation process and skip to later sections that discuss system alteration for nonstandard disk systems.

To achieve device independence, CP/M is separated into three distinct modules:

BIOS	basic I/O system, which is environment dependent
BDOS	basic disk operating system, which is not dependent upon the hardware configuration
CCP	the console command processor, which uses the BDOS

Of these modules, only the BIOS is dependent upon the particular hardware. That is, the user can "patch" the distribution version of CP/M to provide a new BIOS that provides a customized interface between the remaining CP/M modules and the user's own hardware system. This document provides a step-by-step procedure for patching a new BIOS into CP/M.

All disk-dependent portions of CP/M 2 are placed into a BIOS, a resident "disk parameter block," which is either hand coded or produced automatically using the disk definition macro library provided with CP/M 2. The end user need only specify the maximum number of active disks, the starting and ending sector numbers, the data allocation size, the maximum extent of the logical disk, directory size information, and reserved track values. The macros use this information to generate the appropriate tables and table references for use during CP/M 2 operation. Deblocking information is provided, which

aids in assembly or disassembly of sector sizes that are multiples of the fundamental 128 byte data unit, and the system alteration manual includes general purpose subroutines that use the deblocking information to take advantage of larger sector sizes. Use of these subroutines, together with the table-drive data access algorithms, makes CP/M 2 a universal data management system.

File expansion is achieved by providing up to 512 logical file extents, where each logical extent contains 16K bytes of data. CP/M 2 is structured, however, so that as much as 128K bytes of data are addressed by a single physical extent (corresponding to a single directory entry) maintaining compatibility with previous versions while taking advantage of directory space.

If CP/M is being tailored to a computer system for the first time, the new BIOS requires some simple software development and testing. The standard BIOS is listed in Appendix A and can be used as a model for the customized package. A skeletal version of the BIOS given in Appendix B can serve as the basis for a modified BIOS. In addition to the BIOS, the user must write a simple memory loader, called GETSYS, that brings the operating system into memory. To patch the new BIOS into CP/M, the user must write the reverse of GETSYS, called PUTSYS, which places an altered version of CP/M back onto the diskette. PUTSYS can be derived from GETSYS by changing the disk read commands into disk write commands. Sample skeletal GETSYS and PUTSYS programs are described in Section 6.4 and listed in Appendix C. To make the CP/M system load automatically, the user must also supply a cold start loader, similar to the one provided with CP/M (listed in Appendices A and D). A skeletal form of a cold start loader is given in Appendix E, which serves as a model for the loader.

6.2 First Level System Regeneration

The procedure to patch the CP/M system is given below. Address references in each step are shown with "H" denoting the hexadecimal radix, and are given for a 20K CP/M system. For larger CP/M systems, a "bias" is added to each address that is shown with a "+b" following it, where b is equal to the memory size—20K. Values for b in various standard memory sizes are

24K:	b = 24K - 20K = 4K = 1000H
32K:	b = 32K - 20K = 12K = 3000H
40K:	b = 40K - 20K = 20K = 5000H
48K:	b = 48K - 20K = 28K = 7000H
56K:	b = 56K - 20K = 36K = 9000H
62K:	b = 62K - 20K = 42K = A800H
64K:	b = 64K - 20K = 44K = B000H

It should be noted that the standard distribution version of CP/M is set for operation within a 20K memory system. Therefore, the user must first bring up the 20K CP/M system, then configure it for actual memory size (the user should see Section 6.3).

The user should:

1. Read Section 6.4 and write a GETSYS program that reads the first two tracks of a diskette into memory. The program from the diskette must be loaded starting at location 3380H. GETSYS is coded to start at location 100H (base of the TPA), as shown in Appendix C.

2. Test the GETSYS program by reading a blank diskette into memory and check to see that the data have been read properly and that the diskette has not been altered in any way by the GETSYS program.

3. Run the GETSYS program using an initialized CP/M diskette to see if GETSYS loads CP/M starting at 3380H (the operating system actually starts 128 bytes later at 3400H).

4. Read Section 6.4 and write the PUTSYS program. This writes memory starting at 3380H back onto the first two tracks of the diskette. The PUTSYS program should be located at 200H, as shown in Appendix C.

5. Test the PUTSYS program using a blank, uninitialized diskette by writing a portion of memory to the first two tracks; clear memory and read it back using GETSYS. Test PUTSYS completely, since this program will be used to alter CP/M on disk.

6. Study Sections 6.5, 6.6, and 6.7 along with the distribution version of the BIOS given in Appendix A and write a simple version that performs a similar function for the customized environment. Use the program given in Appendix B as a model. Call this new BIOS by the name CBIOS (customized BIOS). Implement only the primitive disk operations on a single drive and simple console input/output functions in this phase.

7. Test CBIOS completely to ensure that it properly performs console character I/O and disk reads and writes. Be careful to ensure that no disk write operations occur during read operations and check that the proper track and sectors are addressed on all reads and writes. Failure to make these checks may cause destruction of the initialized CP/M system after it is patched.

8. Referring to the table in Section 6.5, note that the BIOS is placed between locations 4AOOH and 4FFFH. Read the CP/M system using GETSYS and replace the BIOS segment by the CBIOS developed in step 6 and tested in step 7. This replacement is done in memory.

9. Use PUTSYS to place the patched memory image of CP/M onto the first two tracks of a blank diskette for testing.

10. Use GETSYS to bring the copied memory image from the test diskette back into memory at 3380H and check to ensure that it has loaded back properly (clear memory, if possible, before the load). Upon successful load, branch to the cold start code at location 4A00H. The cold start routine will initialize page zero, then jump to the CCP at location 3400H, which will call the BDOS, which will call the CBIOS. The CBIOS will be asked by the CCP to read sixteen sectors on track 2, and CP/M will type "A>", the system prompt.

If difficulties are encountered, use whatever debug facilities are available to trace and breakpoint the CBIOS.

11. Upon completion of step 10, CP/M has prompted the console for a command input. Test the disk write operation by typing

```
SAVE 1 X.COM
```

(All commands must be followed by a carriage return.) CP/M responds with another prompt (after several disk accesses)

```
A>
```

If it does not, debug the disk write functions and retry.

12. Test the directory command by typing

```
DIR
```

CP/M responds with

```
A: X       COM
```

13. Test the erase command by typing

```
ERA X.COM
```

CP/M responds with the A prompt. This is now an operational system that only requires a bootstrap loader to function completely.

14. Write a bootstrap loader that is similar to GETSYS and place it on track 0, sector 1 using PUTSYS (again using the test diskette, not the distribution diskette). See Sections 6.5 and 6.8 for more information on the bootstrap operation.

15. Retest the new test diskette with the bootstrap loader installed by executing steps 11, 12, and 13. Upon completion of these tests, type a control-C (control and C keys simultaneously). The system executes a "warm start" that reboots the system, and types the A prompt.

16. At this point, there is probably a good version of the customized CP/M system on the test diskette. Use GETSYS to load CP/M from the test diskette. Remove the test diskette, place the distribution diskette (or a legal copy) into the drive, and use PUTSYS to replace the distribution version with the customized version. The user should not make this replacement if unsure of the patch because this step destroys the system that was obtained from Digital Research.

17. Load the modified CP/M system and test it by typing

```
DIR
```

CP/M responds with a list of files that are provided on the initialized diskette. One file is the memory image for the debugger

```
DDT.COM
```

Note that from now on, it is important always to reboot the CP/M system (ctl-C is sufficient) when the diskette is removed and replaced by another diskette, unless the new diskette is to be read only.

18. Load and test the debugger by typing

```
DDT
```

(See Chapter 4 for operating procedures.)

19. Before making further CBIOS modifications, practice using the editor (see Chapter 2), and assembler (see Chapter 3). Recode and test the GETSYS, PUTSYS, and CBIOS programs using ED, ASM, and DDT. Code and test a COPY program that does a sector-to-sector copy from one diskette to another to obtain back-up copies of the original diskette. (Read the CP/M Licensing Agreement specifying legal responsibilities when copying the CP/M system.) Place the copyright notice

Copyright ©, 1979
Digital Research

on each copy that is made with the COPY program.

20. Modify the CBIOS to include the extra functions for punches, readers, and sign-on messages, and add the facilities for additional disk drives, if desired. These changes can be made with the GETSYS and PUTSYS programs or by referring to the regeneration process in Section 6.3.

The user should now have a good copy of the customized CP/M system. Although the CBIOS portion of CP/M belongs to the user, the modified version cannot be legally copied for anyone else's use.

It should be noted that the system remains file-compatible with all other CP/M systems (assuming media compatibility), which allows transfer of nonproprietary software between CP/M users.

6.3 Second Level System Generation

Once the system is running, the user will want to configure CP/M for the desired memory size. Usually a memory image is first produced with the "MOVCPM" program (system relocator) and then placed into a named disk file. The disk file can then be loaded, examined, patched, and replaced using the debugger and the system generation program. (The user should refer to Chapter 1.)

The CBIOS and BOOT are modified using ED and assembled using ASM, producing files called CBIOS.HEX and BOOT. HEX, which contain the code for CBIOS and BOOT in Intel hex format.

To get the memory image of CP/M into the TPA configured for the desired memory size, the user should type the command

```
MOVCPM xx *
```

where xx is the memory size in decimal K bytes (e.g., 32 for 32K). The response will be

```
CONSTRUCTING xxK CP/M VERS 2.0
READY FOR "SYSGEN" OR
"SAVE 34 CPMxx.COM"
```

An image of CP/M in the TPA is configured for the requested memory size. The memory image is at location 0900H through 227FH (i.e., the BOOT is at 0900H, the CCP is at 980H, the BDOS starts at 1180H, and the BIOS is at 1F80H). The user should note that the memory image has the standard MDS-800 BIOS and BOOT on it. It is now necessary to save the memory image in a file so that the user can patch the CBIOS and CBOOT into it:

```
SAVE 34 CPMxx.COM
```

The memory image created by the "MOVCPM" program is offset by a negative bias so that it loads into the free area of the TPA, and thus does not interfere with the operation of CP/M in higher memory. This memory image can be subsequently loaded under DDT and examined or changed in preparation for a new generation of the system. DDT is loaded with the memory image by typing:

```
DDT CPMxx.COM      Load DDT, then read the CPM image.
```

DDT should respond with

```
NEXT  PC
2300  0100
                   (The DDT prompt)
```

The user can then give the display and disassembly commands to examine portions of the memory image between 900H and 227FH. The user should note, however, that to find any particular address within the memory image, one must apply the negative bias to the CP/M address to find the actual address. Track 00, sector 01, is loaded to location 900H (the user should find the cold start loader at 900H to 97FH); track 00, sector 02, is loaded into 980H (this is the base of the CCP); and so on through the entire CP/M system load. In a 20K system, for example, the CCP resides at the CP/M address 3400H, but is placed into memory at 980H by the SYSGEN program. Thus, the negative bias, denoted by n, satisfies

```
3400H + n = 980H, or n = 980H - 3400H
```

Assuming that twos complement arithmetic, n = D580H, which can be checked by

```
3400H + D580H = 10980H = 0980H (ignoring high-order overflow).
```

Note that for larger systems, n satisfies

```
(3400H+b) + n = 980H, or
n = 980H - (3400H + b), or
n = D580H - b
```

The value of n for common CP/M systems is given below.

Memory Size	Bias b	Negative Offset n
20K	0000H	D580H - 0000H = D580H
24K	1000H	D580H - 1000H = C580H
32K	3000H	D580H - 3000H = A580H
40K	5000H	D580H - 5000H = 8580H
48K	7000H	D580H - 7000H = 6580H
56K	9000H	D580H - 9000H = 4580H
62K	A800H	D580H - A800H = 2D80H
64K	B000H	D580H - B000H = 2580H

If the user wants to locate the address x within the memory image loaded under DDT in a 20K system, first type

```
Hx,n          Hexadecimal sum and difference
```

and DDT will respond with the value of x+n (sum) and x-n (difference). The first number printed by DDT is the actual memory address in the image where the data or code are located. The DDT command

```
H3400,D580
```

for example, will produce 980H as the sum, which is where the CCP is located in the memory image under DDT.

The user should type the L command to disassemble portions of the BIOS located at (4A00H+b)-n, which, when one uses the H command, produces an actual address of 1F80H. The disassembly command would thus be

```
L1F80
```

It is now necessary to patch in the CBOOT and CBIOS routines. The BOOT resides at location 0900H in the memory image. If the actual load address is "n", then to calculate the bias (m), the user types the command

```
H900,n          Subtract load address from target address.
```

The second number typed by DDT in response to the command is the desired bias (m). For example, if the BOOT executes at 0080H, the command

```
H900,80
```

will produce

```
0980 0880       Sum and difference in hex.
```

Therefore, the bias "m" would be 0880H. To read-in the BOOT, the user should give the command

```
ICBOOT.HEX      Input file CBOOT.HEX.
```

Then

```
Rm              Read CBOOT with a bias of m (=900H-n).
```

The user may now examine the CBOOT with

```
L900
```

The user is now ready to replace the CBIOS by examining the area at 1F80H where the original version of the CBIOS resides and then typing

```
ICBIOS.HEX      Ready the hex file for loading.
```

The user assumes that the CBIOS is being integrated into a 20K CP/M system and thus originates at location 4A00H. To locate the CBIOS properly in the memory image under DDT, one must apply the negative bias n for a 20K system when loading the hex file. This is accomplished by typing

```
RD580           Read the file with bias D580H.
```

Upon completion of the read, the user should reexamine the area where the CBIOS has been loaded (use an "L1F80" command) to ensure that it was loaded properly. When satisfied that the change has been made, the user should return from DDT using a control-C or, "G0" command.

SYSGEN is used to replace the patched memory image back onto a diskette (the user should utilize a test diskette until sure of the patch), as shown in the following interaction:

SYSGEN	Start the SYSGEN program
SYSGEN VERSION 2.0	Sign-on message from SYSGEN
SOURCE DRIVE NAME (OR RETURN TO SKIP)	Respond with a carriage return to skip the CP/M read operation since the system is already in memory
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)	Respond with "B" to write the new system to the diskette in drive B
DESTINATION ON B, THEN TYPE RETURN	Place a scratch diskette in drive B, then type return.
FUNCTION COMPLETE DESTINATION DRIVE NAME (OR RETURN TO REBOOT)	

The user should place the scratch diskette in drive A and then perform a cold start to bring up the newly configured CP/M system.

The new CP/M system is then tested and the Digital Research copyright notice is placed on the diskette, as specified in the Licensing Agreement:

Copyright ©, 1979
Digital Research

6.4 Sample GETSYS and PUTSYS Programs

The following program provides a framework for the GETSYS and PUTSYS programs referenced in Sections 6.1 and 6.2. The READSEC and WRITESEC subroutines must be inserted by the user to read and write the specific sectors.

```
;   GETSYS PROGRAM — READ TRACKS 0 AND 1 TO MEMORY AT 3380H
;   REGISTER                    USE
;           A                   (SCRATCH REGISTER)
;           B                   TRACK COUNT (0, 1)
;           C                   SECTOR COUNT (1,2,. . .,26)
;           DE                  (SCRATCH REGISTER PAIR)
;           HL                  LOAD ADDRESS
;           SP                  SET TO STACK ADDRESS
;
START:  LXI  SP,3380H       ;SET STACK POINTER TO SCRATCH
                            ;AREA
        LXI  H,.3380H       ;SET BASE LOAD ADDRESS
        MVI  B, 0           ;START WITH TRACK 0
```

```
RDTRK:                          ;READ NEXT TRACK (INITIALLY 0)
        MVI  C,1                ;READ STARTING WITH SECTOR 1
RDSEC:                          ;READ NEXT SECTOR
        CALL  READSEC           ;USER-SUPPLIED SUBROUTINE
        LXI   D,128             ;MOVE LOAD ADDRESS TO NEXT 1/2
                                ;PAGE
        DAD  D                  ;HL = HL + 128
        INR  C                  ;SECTOR = SECTOR + 1
        MOV  A,C                ;CHECK FOR END OF TRACK
        CPI  27
        JC   RDSEC              ;CARRY GENERATED IF SECTOR < 27
;
;   ARRIVE HERE AT END OF TRACK, MOVE TO NEXT TRACK
        INR  B
        MOV  A,B                ;TEST FOR LAST TRACK
        CPI  2
        JC   RDTRK              ;CARRY GENERATED IF TRACK < 2
;
;   ARRIVE HERE AT END OF LOAD, HALT FOR NOW
        HLT
;
;   USER-SUPPLIED SUBROUTINE TO READ THE DISK
READSEC:
;   ENTER WITH TRACK NUMBER IN REGISTER B,
;       SECTOR NUMBER IN REGISTER C, AND
;       ADDRESS TO FILL IN HL
;
        PUSH  B                 ;SAVE B AND C REGISTERS
        PUSH  H                 ;SAVE HL REGISTERS
        ..........................................................
        perform disk read at this point, branch to

        label START if an error occurs
        ..........................................................
        POP H                   ;RECOVER HL
        POP B                   ;RECOVER B AND C REGISTERS
        RET                     ;BACK TO MAIN PROGRAM

        END START
```

This program is assembled and listed in Appendix B for reference purposes, with an assumed origin of 100H. The hexadecimal operation codes that are listed on the left may be useful if the program has to be entered through the panel switches.

The PUTSYS program can be constructed from GETSYS by changing only a few operations in the GETSYS program given above, as shown in Appendix C. The register pair HL becomes the dump address (next address to write), and operations upon these registers do not change within the program. The READSEC subroutine is replaced by a WRITESEC subroutine, which performs the opposite function: data from address HL are written to the track given by register B and sector given by register C. It is often useful to combine GETSYS and PUTSYS into a single program during the test and development phase, as shown in Appendix C.

6.5 Diskette Organization

The sector allocation for the standard distribution version of CP/M is given here for reference purposes. The first sector (see the table on the following page) contains an optional software boot section. Disk controllers are often set up to bring track 0, sector 1, into memory at a specific location (often location 0000H). The program in this sector, called BOOT, has the responsibility of bringing the remaining sectors into memory starting at location 3400H+b. If the user's controller does not have a built-in sector load, the program in track 0, sector 1 can be ignored. In this case, load the program from track 0, sector 2, to location 3400H+b.

As an example, the Intel MDS-800 hardware cold start loader brings track 0, sector 1, into absolute address 3000H. Upon loading this sector, control transfers to location 3000H, where the bootstrap operation commences by loading the remainder of track 0 and all of track 1 into memory, starting at 3400H+b. The user should note that this bootstrap loader is of little use in a non-MDS environment, although it is useful to examine it since some of the boot actions will have to be duplicated in the user's cold start loader.

Track#	Sector#	Page#	Memory Address	CP/M Module name
00	01		(boot address)	Cold Start Loader
00	02	00	3400H+b	CCP
'	03	'	3480H+b	'
'	04	01	3500H+b	'
'	05	'	3580H+b	'
'	06	02	3600H+b	'
'	07	'	3680H+b	'
'	08	03	3700H+b	'
'	09	'	3780H+b	'
'	10	04	3800H+b	'
'	11	'	3880H+b	'
'	12	05	3900H+b	'
'	13	'	3980H+b	'
'	14	06	3A00H+b	'
'	15	'	3A80H+b	'
'	16	07	3B00H+b	'
00	17	'	3B80H+b	CCP
00	18	08	3C00H+b	BDOS
'	19	'	3C80H+b	'
'	20	09	3D00H+b	'
'	21	'	3D80H+b	'
'	22	10	3E00H+b	'
'	23	'	3E80H+b	
'	24	11	3F00H+b	'
'	25	'	3F80H+b	'
'	26	12	4000H+b	'
01	01	'	4080H+b	'
'	02	13	4100H+b	'
'	03	'	4180H+b	'
'	04	14	4200H+b	'
'	05	'	4280H+b	'
'	06	15	4300H+b	'
'	07	'	4380H+b	
'	08	16	4400H+b	'
'	09	'	4480H+b	'

'	10	17	4500H+b	'
'	11	'	4580H+b	'
'	12	18	4600H+b	'
'	13	'	4680H+b	'
'	14	19	4700H+b	'
'	15	'	4780H+b	'
'	16	20	4800H+b	'
'	17	'	4880H+b	'
'	18	21	4900H+b	'
01	19	'	4980H+b	BDOS
07	20	22	4A00H+b	BIOS
'	21	'	4A80H+b	'
'	22	23	4B00H+b	'
'	23	'	4B80H+b	'
'	24	24	4C00H+b	'
01	25	'	4C80H+b	BIOS
01	26	25	4D00H+b	BIOS
02-76	01-26			(directory and data)

6.6 The BIOS Entry Points

The entry points into the BIOS from the cold start loader and BDOS are detailed below. Entry to the BIOS is through a "jump vector" located at 4A00H+b, as shown below (see Appendices A and B, as well). The jump vector is a sequence of 17 jump instructions that send program control to the individual BIOS subroutines. The BIOS subroutines may be empty for certain functions (i.e., they may contain a single RET operation) during reconfiguration of CP/M, but the entries must be present in the jump vector.

The jump vector at 4A00H+b takes the form shown below, where the individual jump addresses are given to the left:

```
4A00H+b   JMP BOOT      ; ARRIVE HERE FROM COLD
                          START LOAD
4A03H+b   JMP WBOOT     ; ARRIVE HERE FOR WARM START
4A06H+b   JMP CONST     ; CHECK FOR CONSOLE CHAR
                          READY
4A09H+b   JMP CONIN     ; READ CONSOLE CHARACTER IN
4A0CH+b   JMP CONOUT    ; WRITE CONSOLE CHARACTER
                          OUT
4A0FH+b   JMP LIST      ; WRITE LISTING CHARACTER OUT
4A12H+b   JMP PUNCH     ; WRITE CHARACTER TO PUNCH
                          DEVICE
4A15H+b   JMP READER    ; READ READER DEVICE
4A18H+b   JMP HOME      ; MOVE TO TRACK 00 ON
                          SELECTED DISK
4A1BH+b   JMP SELDSK    ; SELECT DISK DRIVE
4A1EH+b   JMP SETTRK    ; SET TRACK NUMBER
4A21H+b   JMP SETSEC    ; SET SECTOR NUMBER
```

```
4A24H+b     JMP SETDMA      ; SET DMA ADDRESS
4A27H+b     JMP READ        ; READ SELECTED SECTOR
4A2AH+b     JMP WRITE       ; WRITE SELECTED SECTOR
4A2DH+b     JMP LISTST      ; RETURN LIST STATUS
4A30H+b     JMP SECTRAN     ; SECTOR TRANSLATE
                              SUBROUTINE
```

Each jump address corresponds to a particular subroutine that performs the specific function, as outlined below. There are three major divisions in the jump table: the system (re)initialization, which results from calls on BOOT and WBOOT; simple character I/O performed by calls on CONST, CONIN, CONOUT, LIST, PUNCH, READER, and LISTST; and diskette I/O performed by calls on HOME, SELDSK, SETTRK, SETSEC, SETDMA, READ, WRITE, and SECTRAN.

All simple character I/O operations are assumed to be performed in ASCII, upper and lower case, with high order (parity bit) set to zero. An end-of-file condition for an input device is given by an ASCII control-z (1AH). Peripheral devices are seen by CP/M as "logical" devices and are assigned to physical devices within the BIOS.

To operate, the BDOS needs only the CONST, CONIN, and CONOUT subroutines (LIST, PUNCH, and READER may be used by PIP, but not the BDOS). Further, the LISTST entry is currently used only by DESPOOL, the print spooling utility. Thus, the initial version of CBIOS may have empty subroutines for the remaining ASCII devices.

The characteristics of each device are

CONSOLE	The principal interactive console that communicates with the operator, accessed through CONST, CONIN, and CONOUT; typically, the CONSOLE is a device such as a CRT or teletype.
LIST	The principal listing device, if it exists on the user's system, is usually a hard-copy device, such as a printer or teletype.
PUNCH	The principal tape punching device, if it exists, is normally a high-speed paper tape punch or teletype.
READER	The principal tape reading device, such as a simple optical reader or teletype.

A single peripheral can be assigned as the LIST, PUNCH, and READER device simultaneously. If no peripheral device is assigned as the LIST, PUNCH, or READER device, the CBIOS created by the user may give an appropriate error message so that the system does not "hang" if the device is accessed by PIP or some other user program. Alternately, the PUNCH and LIST routines can just simply return, and the READER routine can return with a 1AH (ctl-Z) in register A to indicate immediate end-of-file.

For added flexibility, the user can optionally implement the "IOBYTE" function, which allows reassignment of physical and logical devices. The IOBYTE function creates a mapping of logical to physical devices that can be altered during CP/M processing (the user should see the STAT command). The definition of the IOBYTE function corresponds to the Intel standard as follows: a single location in memory (currently location 0003H) is maintained, called IOBYTE, which defines the logical to physical device mapping that is in effect at a particular time. The mapping is performed by splitting the

IOBYTE into four distinct fields of two bits each, called the CONSOLE, READER, PUNCH, and LIST fields, as shown below.

	most significant			least significant
IOBYTE AT 003H	LIST	PUNCH	READER	CONSOLE
	bits 6, 7	bits 4, 5	bits 2, 3	bits 0, 1

The value in each field can be in the range 0-3, defining the assigned source or destination of each logical device. The values that can be assigned to each field are given below

CONSOLE field (bits 0,1)

0 console is assigned to the console printer device (TTY:)
1 console is assigned to the CRT device (CRT:)
2 batch mode: use the READER as the CONSOLE input, and the LIST device as the CONSOLE output (BAT:)
3 user defined console device (UC1:)

READER field (bits 2,3)

0 READER is the teletype device (TTY:)
1 READER is the high speed reader device (PTR:)
2 user defined reader # 1 (UR1:)
3 user defined reader # 2 (UR2:)

PUNCH field (bits 4,5)

0 PUNCH is the teletype device (TTY:)
1 PUNCH is the high speed punch device (PTP:)
2 user defined punch # 1 (UP1:)
3 user defined punch # 2 (UP2:)

LIST field (bits 6,7)

0 LIST is the teletype device (TTY:)
1 LIST is the CRT device (CRT:)
2 LIST is the line printer device (LPT:)
3 user defined list device (UL1:)

The implementation of the IOBYTE is optional and affects only the organization of the CBIOS. No CP/M systems use the IOBYTE (although they tolerate the existence of the IOBYTE at location 0003H), except for PIP, which allows access to the physical devices, and STAT, which allows logical-physical assignments to be made or displayed (for more information, the user should see Chapter 1). In any case the IOBYTE implementation should be omitted until the basic CBIOS is fully implemented and tested; then the user should add the IOBYTE to increase the facilities.

Disk I/O is always performed through a sequence of calls on the various disk access subroutines that set up the disk number to access, the track and sector on a particular disk, and the direct memory access (DMA) address involved in the I/O operation. After all these parameters have been set up, a call is made to the READ or WRITE function to perform the actual I/O operation. There is often a single call to SELDSK to select a disk drive, followed by a number of read or write operations to the selected disk before selecting another drive for subsequent operations. Similarly, there may be a single call to set the DMA address, followed by several calls that read or write from the selected DMA address before the DMA address is changed. The track and sector subroutines are always called before the READ or WRITE operations are performed.

The READ and WRITE routines should perform several retries (10 is standard) before reporting the error condition to the BDOS. If the error condition is returned to the BDOS, it will report the error to the user. The HOME subroutine may or may not

actually perform the track 00 seek, depending upon controller characteristics; the important point is that track 00 has been selected for the next operation and is often treated in exactly the same manner as SETTRK with a parameter of 00.

The exact responsibilities of each entry point subroutine are given below.

BOOT The BOOT entry point gets control from the cold start loader and is responsible for basic system initialization, including sending a sign-on message (which can be omitted in the first version). If the IOBYTE function is implemented, it must be set at this point. The various system parameters that are set by the WBOOT entry point must be initialized, and control is transferred to the CCP at 3400+b for further processing. Note that register C must be set to zero to select drive A.

WBOOT The WBOOT entry point gets control when a warm start occurs. A warm start is performed whenever a user program branches to location 0000H, or when the CPU is reset from the front panel. The CP/M system must be loaded from the first two tracks of drive A up to, but not including, the BIOS (or CBIOS, if the user has completed the patch). System parameters must be initialized as shown below:

location 0,1,2	Set to JMP WBOOT for warm starts (000H: JMP 4A03H+b)
location 3	Set initial value of IOBYTE, if implemented in the CBIOS
location 4	High nibble = current user no; low nibble = current drive
location 5,6,7	Set to JMP BDOS, which is the primary entry point to CP/M for transient programs. (0005H: JMP 3C06H+b)

(The user should refer to Section 6.9 for complete details of page zero use.) Upon completion of the initialization, the WBOOT program must branch to the CCP at 3400H+b to (re)start the system. Upon entry to the CCP, register C is set to the drive to select after system initialization. The WBOOT routine should read location 4 in memory, verify that it is a legal drive, and pass it to the CCP in register C.

CONST The user should sample the status of the currently assigned console device and return 0FFH in register A if a character is ready to read and 00H in register A if no console characters are ready.

CONIN The next console character is read into register A, and the parity bit is set (high order bit) to zero. If no console character is ready, the user waits until a character is typed before returning.

CONOUT — The user sends the character from register C to the console output device. The character is in ASCII, with high order parity bit set to zero. The user may want to include a time-out on a line feed or carriage return, if the console device requires some time interval at the end of the line (such as a TI Silent 700 terminal). The user can filter out control characters that cause the console device to react in a strange way (a control-z causes the Lear Seigler terminal to clear the screen, for example).

LIST — The user sends the character from register C to the currently assigned listing device. The character is in ASCII with zero parity bit.

PUNCH — The user sends the character from register C to the currently assigned punch device. The character is in ASCII with zero parity.

READER — The user reads the next character from the currently assigned reader device into register A with zero parity (high order bit must be zero); an end-of-file condition is reported by returning an ASCII control-z(1AH).

HOME — The user moves the disk head of the currently selected disk (initially disk A) to the track 00 position. If the controller allows access to the track 0 flag from the drive, the head is stepped until the track 0 flag is detected. If the controller does not support this feature, the HOME call is translated into a call to SETTRK with a parameter of 0.

SELDSK — The user selects the disk drive given by register C for further operations, where register C contains 0 for drive A, 1 for drive B, and so on up to 15 for drive P (the standard CP/M distribution version supports four drives). On each disk select, SELDSK must return in HL the base address of a 16-byte area, called the Disk Parameter Header, described in Section 6.10. For standard floppy disk drives, the contents of the header and associated tables do not change; thus, the program segment included in the sample CBIOS performs this operation automatically. If there is an attempt to select a nonexistent drive, SELDSK returns HL=0000H as an error indicator. Although SELDSK must return the header address on each call, it is advisable to postpone the physical disk select operation until an I/O function (seek, read, or write) is actually performed, since disk selects often occur without utimately performing any disk I/O, and many controllers will unload the head of the current disk before selecting the new drive. This would cause an excessive amount of noise and disk wear. The least significant bit of register E is zero if this is the first occurrence of the drive select since the last cold or warm start.

SETTRK — Register BC contains the track number for subsequent disk accesses on the currently selected drive. The sector number in BC is the same as the number returned from the SECTRAN entry point. The user can choose to seek the selected track at

this time or delay the seek until the next read or write actually occurs. Register BC can take on values in the range 0-76 corresponding to valid track numbers for standard floppy disk drives and 0-65535 for nonstandard disk subsystems.

SETSEC

Register BC contains the sector number (1 through 26) for subsequent disk accesses on the currently selected drive. The sector number in BC is the same as the number returned from the SECTRAN entry point. The user can choose to send this information to the controller at this point or delay sector selection until a read or write operation occurs.

SETDMA

Register BC contains the DMA (disk memory access) address for subsequent read or write operations. For example, if B = 00H and C = 80H when SETDMA is called, all subsequent read operations read their data into 80H through 0FFH and all subsequent write operations get their data from 80H through 0FFH, until the next call to SETDMA occurs. The initial DMA address is assumed to be 80H. The controller need not actually support direct memory access. If, for example, all data transfers are through I/O ports, the CBIOS that is constructed will use the 128-byte area starting at the selected DMA address for the memory buffer during the subsequent read or write operations.

READ

Assuming the drive has been selected, the track has been set, the sector has been set, and the DMA address has been specified, the READ subroutine attempts to read one sector based upon these parameters and returns the following error codes in register A:

0	no errors occurred
1	nonrecoverable error condition occurred

Currently, CP/M responds only to a zero or nonzero value as the return code. That is, if the value in register A is 0, CP/M assumes that the disk operation was completed properly. If an error occurs, however, the CBIOS should attempt at least 10 retries to see if the error is recoverable. When an error is reported the BDOS will print the message "BDOS ERR ON x: BAD SECTOR". The operator then has the option of typing carriage-return to ignore the error, or ctl-C to abort.

WRITE

The user writes the data from the currently selected DMA address to the currently selected drive, track, and sector. For floppy disks, the data should be marked as "nondeleted data" to maintain compatibility with other CP/M systems. The error codes given in the READ command are returned in register A, with error recovery attempts as described above.

LISTST

The user returns the ready status of the list device used by the DESPOOL program to improve console response during its operation. The value 00 is returned in A if the list device is not ready to accept a character and 0FFH if a character can be sent

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

to the printer. A 00 value should be returned if LIST status is not implemented.

SECTRAN — The user performs logical to physical sector translation to improve the overall response of CP/M. Standard CP/M systems are shipped with a "skew factor" of 6, where six physical sectors are skipped between each logical read operation. This skew factor allows enough time between sectors for most programs to load their buffers without missing the next sector. In particular computer systems that use fast processors, memory, and disk subsystems, the skew factor may be changed to improve overall response. However, the user should mtaintain a single density IBM-compatible version of CP/M for information transfer into and out of the computer system, using a skew factor of 6. In general, SECTRAN receives a logical sector number relative to zero in BC and a translate table address in DE. The sector number is used as an index into the translate table, with the resulting physical sector number in HL. For standard systems, the table and indexing code is provided in the CBIOS and need not be changed.

6.7 A Sample BIOS

The program shown in Appendix B can serve as a basis for a user's first BIOS. The simplest functions are assumed in this BIOS, so that the user can enter it through a front panel, if absolutely necessary. The user must alter and insert code into the subroutines for CONST, CONIN, CONOUT, READ, WRITE, and WAITIO subroutines. Storage is reserved for user-supplied code in these regions. The scratch area reserved in page zero (see section 6.9) for the BIOS is used in this program, so that it could be implemented in ROM, if desired.

Once operational, this skeletal version can be enhanced to print the initial sign-on message and perform better error recovery. The subroutines for LIST, PUNCH, and READER can be filled out and the IOBYTE function can be implemented.

6.8 A Sample Cold Start Loader

The program shown in Appendix E can serve as a basis for a cold start loader. The disk read function must be supplied by the user, and the program must be loaded somehow starting at location 0000. Space is reserved for the patch code so that the total amount of storage required for the cold start loader is 128 bytes. Eventually, the user will probably want to get this loader onto the first disk sector (track 0, sector 1) and cause the controller to load it into memory automatically upon system start up. Alternatively, the cold start loader can be placed into ROM, and above the CP/M system. In this case, it will be necessary to originate the program at a higher address and key in a jump instruction at system start up that branches to the loader. Subsequent warm starts will not require this key-in operation, since the entry point WBOOT gets control thus bringing the system in from disk automatically. The skeletal cold start loader has minimal error recover, which may be enhanced in later versions.

6.9 Reserved Locations in Page Zero

Main memory page zero, between locations 00H and 0FFH, contains several segments of code and data that are used during CP/M processing. The code and data areas are given below for reference

Locations from to	Contents
0000H-0002H	Contains a jump instruction to the warm start entry point at location 4A03H+b. This allows a simple programmed restart (JMP 0000H) or manual restart from the front panel.
0003H-0003H	Contains the Intel standard IOBYTE, which is optionally included in the user's CBIOS, as described in Section 6.6.
0004H-0004H	Current default drive number (0=A,...,15=P).
0005H-0007H	Contains a jump instruction to the BDOS and serves two purposes: JMP 0005H provides the primary entry point to the BDOS, as described in Chapter 5, and LHLD 0006H brings the address field of the instruction to the HL register pair. This value is the lowest address in memory used by CP/M (assuming the CCP is being overlaid). The DDT program will change the address field to reflect the reduced memory size in debug mode.
0008H-0027H	(Interrupt locations 1 through 5 not used.)
0030H-0037H	(Interrupt location 6, not currently used; reserved.)
0038H-003AH	Restart 7; contains a jump instruction into the DDT or SID program when running in debug mode for programmed breakpoints, but is not otherwise used by CP/M.
003BH-003FH	(Not currently used; reserved.)
0040H-004FH	A 16-byte area reserved for scratch by CBIOS, but is not used for any purpose in the distribution version of CP/M.
0050H-005BH	(Not currently used; reserved.)
005CH-007CH	Default file control block produced for a transient program by the Console Command Processor.
007DH-007FH	Optional default random record position.
0080H-00FFH	Default 128-byte disk buffer (also filled with the command line when a transient is loaded under the CCP).

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

This information is set up for normal operation under the CP/M system, but can be overwritten by a transient program if the BDOS facilities are not required by the transient.

If, for example, a particular program performs only simple I/O and must begin execution at location 0, it can first be loaded into the TPA, using normal CP/M facilities, with a small memory move program that gets control when loaded (the memory move program must get control from location 0100H, which is the assumed beginning of all transient programs). The move program can then proceed to move the entire memory image down to location 0 and pass control to the starting address of the memory load. If the BIOS is overwritten or if location 0 (containing the warm start entry point) is overwritten, the operator must bring the CP/M system back into memory with a cold start sequence.

6.10 Disk Parameter Tables

Tables are included in the BIOS that describe the particular characteristics of the disk subsystem used with CP/M. These tables can be either hand-coded, as shown in the sample CBIOS in Appendix B, or automatically generated using the DISKDEF macro library, as shown in Appendix F. The purpose here is to describe the elements of these tables.

In general, each disk drive has an associated (16-byte) disk parameter header that contains information about the disk drive and provides a scratchpad area for certain BDOS operations. The format of the disk parameter header for each drive is shown below.

Disk Parameter Header

XLT	0000	0000	0000	DIRBUF	DPB	CSV	ALV
16b	16b	16b	16b	16b	16b	16b	16b

where each element is a word (16-bit) value. The meaning of each Disk Parameter Header (DPH) element is

XLT	Address of the logical to physical translation vector, if used for this particular drive, or the value 0000H if no sector translation takes place (i.e., the physical and logical sector numbers are the same). Disk drives with identical sector skew factors share the same translate tables.
0000	Scratchpad values for use within the BDOS (initial value is unimportant).
DIRBUF	Address of a 128-byte scratchpad area for directory operations within BDOS. All DPHs address the same scratchpad area.
DPB	Address of a disk parameter block for this drive. Drives with identical disk characteristics address the same disk parameter block.
CSV	Address of a scratchpad area used for software check for changed disks. This address is different for each DPH.
ALV	Address of a scratchpad area used by the BDOS to keep disk storage allocation information. This address is different for each DPH.

Given n disk drives, the DPHs are arranged in a table whose first row of 16 bytes corresponds to drive 0, with the last row corresponding to drive n-1. The table thus appears as

```
DPBASE:
   00 | XLT 00 | 0000 | 0000 | 0000 | DIRBUF | DBP 00 | CSV 00 | ALV 00 |
   01 | XLT 01 | 0000 | 0000 | 0000 | DIRBUF | DBP 01 | CSV 01 | ALV 01 |
                          (and so on through)
  n-1 | XLTn-1 | 0000 | 0000 | 0000 | DIRBUF | DBPn-1 | CSVn-1 | ALVn-1 |
```

where the label DPBASE defines the base address of the DPH table.

A responsibility of the SELDSK subroutine is to return the base address of the DPH for the selected drive. The following sequence of operations returns the table address, with a 0000H returned if the selected drive does not exist.

```
NDISKS      EQU     4           ;NUMBER OF DISK DRIVES
.....
SELDSK:     ;SELECT DISK GIVEN BY BC
            LXI     H,0000H     ;ERROR CODE
            MOV     A,C         ;DRIVE OK?
            CPI     NDISKS      ;CY IF SO
            RNC                 ;RET IF ERROR
            ;NO ERROR, CONTINUE
            MOV     L,C         ;LOW(DISK)
            MOV     H,B         ;HIGH(DISK)
            DAD     H           ;*2
            DAD     H           ;*4
            DAD     H           ;*8
            DAD     H           ;*16
            LXI     D,DPBASE;FIRST DPH
            DAD     D           ;DPH(DISK)
            RET
```

The translation vectors (XLT 00 through XLTn-1) are located elsewhere in the BIOS, and simply correspond one-for-one with the logical sector numbers zero through the sector count 1. The Disk Parameter Block (DPB) for each drive is more complex. A particular DPB, which is addressed by one or more DPHs, takes the general form

SPT	BSH	BLM	EXM	DSM	DRM	AL0	AL1	CKS	OFF
16b	8b	8b	8b	16b	16b	8b	8b	16b	16b

where each is a byte or word value, as shown by the 8b or 16b indicator below the field.

SPT is the total number of sectors per track.

BSH is the data allocation block shift factor, determined by the data block allocation size.

BLM is the data allocation block mask (2[BSH-1]).

EXM is the extent mask, determined by the data block allocation size and the number of disk blocks.

DSM determines the total storage capacity of the disk drive.

DRM	determines the total number of directory entries that can be stored on this drive. (AL0,AL1 determine reserved directory blocks.)
CKS	is the size of the directory check vector.
OFF	is the number of reserved tracks at the beginning of the (logical) disk.

The values of BSH and BLM determine (implicitly) the data allocation size BLS, which is not an entry in the DPB. Given that the designer has selected a value for BLS, the values of BSH and BLM are shown in the tabulation below.

BLS	BSH	BLM
1024	3	7
2048	4	15
4096	5	31
8192	6	63
16384	7	127

where all values are in decimal. The value of EXM depends upon both the BLS and whether the DSM value is less than 256 or greater than 255. For DSM $<$ 256 the value of EXM is given by:

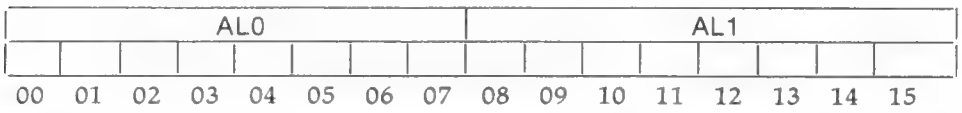
BLS	EXM
1024	0
2048	1
4096	3
8192	7
16384	15

For DSM $>$ 255 the value of EXM is given by:

BLS	EXM
1024	N/A
2048	0
4096	1
8192	3
16384	7

The value of DSM is the maximum data block number supported by this particular drive, measured in BLS units. The product BLS times (DSM+1) is the total number of bytes held by the drive and, of course, must be within the capacity of the physical disk, not counting the reserved operating system tracks.

The DRM entry is the one less than the total number of directory entries that can take on a 16-bit value. The values of AL0 and AL1, however, are determined by DRM. The values AL0 and AL1 can together be considered a string of 16-bits, as shown below.



where position 00 corresponds to the high order bit of the byte labeled AL0 and 15 corresponds to the low order bit of the byte labeled AL1. Each bit position reserves a data block for number of directory entries, thus allowing a total of 16 data blocks to be

assigned for directory entries (bits are assigned starting at 00 and filled to the right until position 15). Each directory entry occupies 32 bytes, resulting in the tabulation below.

BLS	Directory Entries
1024	32 times # bits
2048	64 times # bits
4096	128 times # bits
8192	256 times # bits
16384	512 times # bits

Thus, if DRM = 127 (128 directory entries) and BLS = 1024, there are 32 directory entries per block, requiring 4 reserved blocks. In this case, the 4 high order bits of AL0 are set, resulting in the values AL0 = 0F0H and AL1 = 00H.

The CKS value is determined as follows: if the disk drive media is removable, then CKS = (DRM+1)/4, where DRM is the last directory entry number. If the media are fixed, then set CKS = 0 (no directory records are checked in this case).

Finally, the OFF field determines the number of tracks that are skipped at the beginning of the physical disk. This value is automatically added whenever SETTRK is called and can be used as a mechanism for skipping reserved operating system tracks or for partitioning a large disk into smaller segmented sections.

To complete the discussion of the DPB, several DPHs can address the same DPB if their drive characteristics are identical. Further, the DPB can be dynamically changed when a new drive is addressed by simply changing the pointer in the DPH since the BDOS copies the DPB values to a local area whenever the SELDSK function is invoked.

Returning back to the DPH for a particular drive, the two address values CSV and ALV remain. Both addresses reference an area of uninitialized memory following the BIOS. The areas must be unique for each drive, and the size of each area is determined by the values in the DPB.

The size of the area addressed by CSV is CKS bytes, which is sufficient to hold the directory check information for this particular drive. If CKS = (DRM+1)/4, one must reserve (DRM+1)/4 bytes for directory check use. If CKS = 0, no storage is reserved.

The size of the area addressed by ALV is determined by the maximum number of data blocks allowed for this particular disk and is computed as (DSM/8)+1.

The CBIOS shown in Appendix B demonstrates an instance of these tables for standard 8-inch single density drives. It may be useful to examine this program and compare the tabular values with the definitions given above.

6.11 The DISKDEF Macro Library

A macro library is shown in Appendix F, called DISKDEF, which greatly simplifies the table construction process. One must have access to the MAC macro assembler, of course, to use the DISKDEF facility, while the macro library is included with all CP/M 2 distribution disks.

A BIOS disk definition consists of the following sequence of macro statements:

```
MACLIB    DISKDEF
.....
DISKS     n
DISKDEF   0,...
DISKDEF   1,...
```


ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
.....
DISKDEF         n-1
.....
ENDEF
```

where the MACLIB statement loads the DISKDEF.LIB file (on the same disk as the BIOS) into MAC's internal tables. The DISKS macro call follows, which specifies the number of drives to be configured with the user's system, where n is an integer in the range 1 to 16. A series of DISKDEF macro calls then follow that define the characteristics of each logical disk, 0 through n-1 (corresponding to logical drives A through P). The DISKS and DISKDEF macros generate the in-line fixed data tables described in the previous section and thus must be placed in a nonexecutable portion of the BIOS, typically directly following the BIOS jump vector.

The remaining portion of the BIOS is defined following the DISKDEF macros, with the ENDEF macro call immediately preceding the END statement. The ENDEF (End of Diskdef) macro generates the necessary uninitialized RAM areas, which are located in memory above the BIOS.

The form of the DISKDEF macro call is

```
DISKDEF  dn,fsc,lsc,[skf],bls,dks,dir,cks,ofs,[0]
```

where

dn	is the logical disk number, 0 to n-1.
fsc	is the first physical sector number (0 or 1).
lsc	is the last sector number.
skf	is the optional sector skew factor.
bls	is the data allocation block size.
dks	is the number of blocks on the disk.
dir	is the number of directory entries.
cks	is the number of "checked" directory entries.
ofs	is the track offset to logical track 00.
[0]	is an optional 1.4 compatibility flag.

The value dn is the drive number being defined with this DISKDEF macro invocation. The fsc parameter accounts for differing sector numbering systems and is usually 0 or 1. The lsc is the last numbered sector on a track. When present, the skf parameter defines the sector skew factor, which is used to create a sector translation table accodrding to the skew.

If the number of sectors is less than 256, a single-byte table is created, otherwise each translation table element occupies two bytes. No translation table is created if the skf parameter is omitted (or equal to 0). The bls parameter specifies the number of bytes allocated to each data block, and takes on the values 1024, 2048, 4096, 8192, or 16384. Generally, performance increases with larger data block sizes since there are fewer directory references and logically connected data records are physically close on the disk. Further, each directory entry addresses more data and the BIOS-resident ram space is reduced.

The dks parameter specifies the total disk size in bls units. That is, if the bls = 2048 and dks = 1000, the total disk capacity is 2,048,000 bytes. If dks is greater than 255, the block size parameter bls must be greater than 1024. The value of dir is the total number of directory entries, which may exceed 255, if desired. The cks parameter determines the

number of directory items to check on each directory scan and is used internally to detect changed disks during system operation, where an intervening cold or warm start has not occurred (when this situation is detected, CP/M automatically marks the disk read/only so that data are not subsequently destroyed).

As stated in the previous section, the value of cks = dir when the medium is easily changed, as is the case with a floppy disk subsystem. If the disk is permanently mounted, the value of cks is typically 0, since the probability of changing disks without a restart is low. The ofs value determines the number of tracks to skip when this particular drive is addressed, which can be used to reserve additional operating system space or to simulate several logical drives on a single large capacity physical drive. Finally, the [0] parameter is included when file compatibility is required with versions of 1.4 that have been modified for higher density disks. This parameter ensures that only 16K is allocated for each directory record, as was the case for previous versions. Normally, this parameter is not included.

For convenience and economy of table space, the special form

```
DISKDEF     i,j
```

gives disk i the same characteristics as a previously defined drive j. A standard four-drive single density system, which is compatible with version 1.4, is defined using the following macro invocations:

```
DISKS       4
DISKDEF     0,1,26,6,1024,243,64,64,2
DISKDEF     1,0
DISKDEF     2,0
DISKDEF     3,0
....
ENDEF
```

with all disks having the same parameter values of 26 sectors per track (numbered 1 through 26), with 6 sectors skipped between each access, 1024 bytes per data block, 243 data blocks for a total of 243K-byte disk capacity, 64 checked directory entries, and two operating system tracks.

The DISKS macro generates n DPHs, starting at the DPH table address DPBASE generated by the macro. Each disk header block contains sixteen bytes, as described above, and correspond one-for-one to each of the defined drives. In the four-drive standard system, for example, the DISKS macro generates a table of the form:

```
DPBASE      EQU $
DPE0:       DW  XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV0,ALV0
DPE1:       DW  XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV1,ALV1
DPE2:       DW  XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV2,ALV2
DPE3:       DW  XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV3,ALV3
```

where the DPH labels are included for reference purposes to show the beginning table addresses for each drive 0 through 3. The values contained within the DPH are described in detail in the previous section. The check and allocation vector addresses are generated by the ENDEF macro in the ram area following the BIOS code and tables.

The user should note that if the skf (skew factor) parameter is omitted (or equal to 0), the translation table is omitted and a 0000H value is inserted in the XLT position of the DPH for the disk. In a subsequent call to perform the logical to physical translation, SECTRAN receives a translation table address of DE = 0000H and simply returns the original logical sector from BC in the HL register pair. A translate table is constructed when the skf parameter is present, and the (nonzero) table address is placed into the

corresponding DPHs. The tabulation shown below, for example, is constructed when the standard skew factor skf = 6 is specified in the DISKDEF macro call:

```
XLT0:   DB      1,7,13,19,25,5,11,17,23,3,9,15,21
        DB      2,8,14,20,26,6,12,18,24,4,10,16,22
```

Following the ENDEF macro call, a number of uninitialized data areas are defined. These data areas need not be a part of the BIOS that is loaded upon cold start, but must be available between the BIOS and the end of memory. The size of the uninitialized RAM area is determined by EQU statements generated by the ENDEF macro. For a standard four-drive system, the ENDEF macro might produce

```
4C72 =      BEGDAT EQU $
            (data areas)
4DB0 =      ENDDAT EQU $
013C =      DATSIZ EQU $-BEGDAT
```

which indicates that uninitialized RAM begins at location 4C72H, ends at 4DB0H-1, and occupies 013CH bytes. The user must ensure that these addresses are free for use after the system is loaded.

After modification, the user can utilize the STAT program to check drive characteristics, since STAT uses the disk parameter block to decode the drive information. The STAT command form

```
STAT d:DSK:
```

decodes the disk parameter block for drive d (d=A,...,P) and displays the values shown below.

- r: 128-byte record capacity
- k: kilobyte drive capacity
- d: 32-byte directory entries
- c: checked directory entries
- e: records/extent
- b: records/block
- s: sectors/track
- t: reserved tracks

Three examples of DISKDEF macro invocations are shown below with corresponding STAT parameter values (the last produces a full 8-megabyte system).

```
                DISKDEF 0,1,58,,2048,256,128,128,2
r=4096,         k=512, d=128, c=128, e=256, b=16, s=58, t=2

                DISKDEF 0,1,58,,2048,1024,300,0,2
r=16384,        k=2048, d=300, c=0, e=128, b=16, s=58, t=2

                DISKDEF 0,1,58,,16384,512,128,128,2
r=65536,        k=8192, d=128, c=128, e=1024, b=128, s=58, t=2
```

6.12 Sector Blocking and Deblocking

Upon each call to the BIOS WRITE entry point, the CP/M BDOS includes information that allows effective sector blocking and deblocking where the host disk subsystem has a sector size that is a multiple of the basic 128-byte unit. The purpose here is to present a general-purpose algorithm that can be included within the BIOS and that uses the BDOS information to perform the operations automatically.

On each call to WRITE, the BDOS provides the following information in register C:

0	=	normal sector write
1	=	write to directory sector
2	=	write to the first sector of a new data block

Condition 0 occurs whenever the next write operation is into a previously written area, such as a random mode record update, when the write is to other than the first sector of an unallocated block, or when the write is not into the directory area. Condition 1 occurs when a write into the directory area is performed. Condition 2 occurs when the first record (only) of a newly allocated data block is written. In most cases, application programs read or write multiple 128-byte sectors in sequence; thus, there is little overhead involved in either operation when blocking and deblocking records, since preread operations can be avoided when writing records.

Appendix G lists the blocking and deblocking algorithms in skeletal form (this file is included on your CP/M disk). enerally, the algorithms map all CP/M sector read operations onto the host disk through an intermediate buffer that is the size of the host disk sector. Throughout the program, values and variables that relate to the CP/M sector involved in a seek operation are prefixed by sek, while those related to the host disk system are prefixed by hst. The equate statements beginning on line 29 of Appendix G define the mapping between CP/M and the host system, and must be changed if other than the sample host system is involved.

The entry points BOOT and WBOOT must contain the initialization code starting on line 57, while the SELDSK entry point must be augmented by the code starting on line 65. The user should note that although the SELDSK entry point computes and returns the Disk Parameter Header address, it does not physically select the host disk at this point (it is selected later at READHST or WRITEHST). Further, SETTRK, SETTRK, and SETDMA simply store the values, but do not take any other action at this point. SECTRAN performs a trivial function of returning the physical sector number.

The principal entry points are READ and WRITE, starting on lines 110 and 125, respectively. These subroutines take the place of your previous READ and WRITE operations.

The actual physical read or write takes place at either WRITEHST or READHST, where all values have been prepared: hstdsk is the host disk number, hsttrk is the host track number, and hstsec is the host sector number (which may require translation to a physical sector number). The user must insert code at this point that performs the full host sector read or write into or out of the buffer at hstbuf of length hstsiz. All other mapping functions are performed by the algorithms.

This particular algorithm was tested using an 80-megabyte hard disk unit that was originally configured for 128-byte sectors, producing approximately 35 megabytes of formatted storage. When configured for 512-byte host sectors, usable storage increased to 57 megabytes, with a corresponding 400% improvement in overall response. In this situation, there is no apparent overhead involved in deblocking sectors, with the advantage that user programs still maintain 128-byte sectors. This is primarily because of the information provided by the BDOS, which eliminates the necessity for preread operations.

Appendix A: The MDS Basic I/O System (BIOS)

```
                ;       mds-800 i/o drivers for cp/m 2.2
                ;       (four drive single density version)
                ;
                ;       version 2.2 february, 1980
                ;
0016 =          vers    equ     22              ;version 2.2
                ;
                ;       copyright (c) 1980
                ;       digital research
                ;       box 579, pacific grove
                ;       california, 93950
                ;
                ;
ffff =          true    equ     offffh          ;value of "true"
0000 =          false   equ     not true        ;"false"
0000 =          test    equ     false           ; true if test bios
                ;
                        if      test
                bias    equ     03400h          ;base of ccp in test system
                        endif
                        if      not test
0000 =          bias    equ     0000h           ;generate relocatable cp/m system
                        endif
                ;
1600 =          patch   equ     1600h
                ;
1600                    org     patch
0000 =          cpmb    equ     $-patch         ;base of cpm console processor
0806 =          bdos    equ     806h+cpmb       ;basic dos (resident portion)
```

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
30    1600 =          cpml    equ     $-cpmb      ;length (in bytes) of cpm system
31    002c =          nsects  equ     cpml/128    ;number of sectors to load
32    0002 =          offset  equ     2           ;number of disk tracks used by cp/m
33    0004 =          cdisk   equ     0004h       ;address of last logged disk on warm start
34    0080 =          buff    equ     0080h       ;default buffer address
35    000a =          retry   equ     10          ;max retries on disk i/o before error
36                    ;
37                    ;       perform following functions
38                    ;       boot      cold start
39                    ;       wboot  warm start (save i/o byte)
40                    ;       (boot and wboot are the same for mds)
41                    ;       const     console status
42                    ;                 reg-a = 00 if no character ready
43                    ;                 reg-a = ff if character ready
44                    ;       conin     console character in (result in reg-a)
45                    ;       conout    console character out (char in reg-c)
46                    ;       list      list out (char in reg-c)
47                    ;       punch     punch out (char in reg-c)
48                    ;       reader    paper tape reader in (result to reg-a)
49                    ;       home      move to track 00
50                    ;
51                    ;       (the following calls set-up the io parameter block for the
52                    ;       mds, which is used to perform subsequent reads and writes)
53                    ;       seldsk    select disk given by reg-c (0, 1, 2. . . )
54                    ;       settrk    set track address (0, . . . 76) for subsequent read/write
55                    ;       setsec    set sector address (1, . . . , 26) for subsequent read/write
56                    ;       setdma    set subsequent dma address(initially 80h)
57                    ;
58                    ;       (read and write assume previous calls to set up the io parameters)
59                    ;       read      read track/sector to preset dma address
60                    ;       write     track/sector from preset dma address
61                    ;
62                    ;       jump vector for individual routines
63    1600 c3b316             jmp       boot
```

```
64    1603 c3c316          wboote:  jmp     wboot
65    1606 c36117                   jmp     const
66    1609 c36417                   jmp     conin
67    160c c36a17                   jmp     conout
68    160f c36d17                   jmp     list
69    1612 c37217                   jmp     punch
70    1615 c37517                   jmp     reader
71    1618 c37817                   jmp     home
72    161b c37d17                   jmp     seldsk
73    161e c3a717                   jmp     settrk
74    1621 c3ac17                   jmp     setsec
75    1624 c3bb17                   jmp     setdma
76    1627 c3c117                   jmp     read
77    162a c3ca17                   jmp     write
78    162d c37017                   jmp     listst      ;list status
79    1630 c3b117                   jmp     sectran
80                               ;
81                                  maclib  diskdef     ;load the disk definition library
82                                  disks   4           ;four disks
83    1633+=               dpbase   equ     $           ;base of disk parameter blocks
84    1633+82160000        dpe0:    dw      xlt0, 0000h         ;translate table
85    1637+00000000                 dw      0000h, 0000h        ;scratch area
86    163b+6e187316                 dw      dirbuf, dpb0        ;dir buff, parm block
87    163f+0d19ee18                 dw      csv0, alv0          ;check, alloc vectors
88    1643+82160000        dpe1:    dw      xlt1, 0000h         ;translate table
89    1647+00000000                 dw      0000h, 0000h        ;scratch area
90    164b+6e187316                 dw      dirbuf, dpb1        ;dir buff, parm block
91    164f+3c191d19                 dw      csv1, alv1          ;check, alloc vectors
92    1653+82160000        dpe2:    dw      xlt2, 0000h         ;translate table
93    1657+00000000                 dw      0000h, 0000h        ;scratch area
94    165b+6e187316                 dw      dirbuf, dpb2        ;dir buff, parm block
95    165f+6b194c19                 dw      csv2, alv2          ;check, alloc vectors
96    1663+82160000        dpe3:    dw      xlt3, 0000h         ;translate table
97    1667+00000000                 dw      0000h, 0000h        ;scratch area
```

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
 98    166b+6e187316          dw      dirbuf, dpb3        ;check, alloc block
 99    166f+9a197b19          dw      csv3, alv3          ;dir buff, parm vectors
100                          diskdef  0, 1, 26, 6, 1024, 243, 64, 64, offset
101    1673+=          dpb0   equ     $                   ;disk parm block
102    1673+1a00              dw      26                  ;sec per track
103    1675+03                db      3                   ;block shift
104    1676+07                db      7                   ;block mask
105    1677+00                db      0                   ;extnt mask
106    1678+f200              dw      242                 ;disk size-1
107    167a+3f00              dw      63                  ;directory max
108    167c+c0                db      192                 ;alloc0
109    167d+00                db      0                   ;alloc1
110    167e+1000              dw      16                  ;check size
111    1680+0200              dw      2                   ;offset
112    1682+=          xlt0   equ     $                   ;translate table
113    1682+01                db      1
114    1683+07                db      7
115    1684+0d                db      13
116    1685+13                db      19
117    1686+19                db      25
118    1687+05                db      5
119    1688+0b                db      11
120    1689+11                db      17
121    168a+17                db      23
122    168b+03                db      3
123    168c+09                db      9
124    168d+0f                db      15
125    168e+15                db      21
126    168f+02                db      2
127    1690+08                db      8
128    1691+0e                db      14
129    1692+14                db      20
130    1693+1a                db      26
131    1694+06                db      6
```

```
132     1695+0c                 db      12
133     1696+12                 db      18
134     1697+18                 db      24
135     1698+04                 db      4
136     1699+0a                 db      10
137     169a+10                 db      16
138     169b+16                 db      22
139                             diskdef 1, 0
140     1673+=          dpb1    equ     dpb0        ;equivalent parameters
141     001f+=          als1    equ     als0        ;same allocation vector size
142     0010+=          css1    equ     css0        ;same checksum vector size
143     1682+=          xlt1    equ     xlt0        ;same translate table
144                             diskdef 2, 0
145     1673+=          dpb2    equ     dpb0        ;equivalent parameters
146     001f+=          als2    equ     als0        ;same allocation vector size
147     0010+=          css2    equ     css0        ;same checksum vector size
148     1682+=          xlt2    equ     xlt0        ;same translate table
149                             diskdef 3, 0
150     1673+=          dpb3    equ     dpb0        ;equivalent parameters
151     001f+=          als3    equ     als0        ;same allocation vector size
152     0010+=          css3    equ     css0        ;same checksum vector size
153     1682+=          xlt3    equ     xlt0        ;same translate table
154                     ;       endef occurs at end of assembly
155                     ;
156                     ;       end of controller--independent code, the remaining subroutines
157                     ;       are tailored to the particular operating environment, and must
158                     ;       be altered for any system which differs from the intel mds.
159                     ;
160                     ;       the following code assumes the mds monitor exists at 0f800h
161                     ;       and uses the i/o subroutines within the monitor
162                     ;
163                     ;       we also assume the mds system has four disk drives
164     00fd =          revrt   equ     0fdh        ;interrupt revert port
165     00fc =          intc    equ     0fch        ;interrupt mask port
```

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
00f3 =              icon    equ     0f3h        ;interrupt control port
007E =              inte    equ     0111$1110b      ;enable rst 0 (warm boot), rst 7 (monitor)
                    ;
                    ;       mds monitor equates
f800 =              mon80   equ     0f800h      ;mds monitor
ff0f =              rmon80  equ     0ff0fh      ;restart mon80 (boot error)
f803 =              ci      equ     0f803h      ;console character to reg-a
f806 =              ri      equ     0f806h      ;reader in to reg-a
f809 =              co      equ     0f809h      ;console char from c to console out
f80c =              po      equ     0f80ch      ;punch char from c to punch device
f80f =              lo      equ     0f80fh      ;list from c to list device
f812 =              csts    equ     0f812h      ;console status 00/ff to register a
                    ;
                    ;       disk ports and commands
0078 =              base    equ     78h         ;base of disk command io ports
0078 =              dstat   equ     base        ;disk status (input)
0079 =              rtype   equ     base+1      ;result type (input)
007b =              rbyte   equ     base+3      ;result byte (input)
                    ;
0079 =              ilow    equ     base+1      ;iopb low address (output)
007a =              ihigh   equ     base+2      ;iopb high address (output)
                    ;
0004 =              readf   equ     4h          ;read function
0006 =              writf   equ     6h          ;write function
0003 =              recal   equ     3h          ;recalibrate drive
0004 =              iordy   equ     4h          ;i/o finished mask
000d =              cr      equ     0dh         ;carriage return
000a =              lf      equ     0ah         ;line feed
                    ;
                    signon: ;signon message: xxk cp/m vers y.y
169c 0d0a0a                 db      cr, lf, lf
                            if      test
                            db      '32'        ;32k example bios
                            endif
```

```
200                                     if      not test
201     169f 3030                       db      '00'            ;memory size filled by relocator
202                                     endif
203     16a1 6b2043502f                 db      'k cp/m vers '
204     16ad 322e32                     db      vers/10+'0', '.' vers mod 10+'0'
205     16b0 0d0a00                     db      cr, lf, 0
206                             ;
207                             boot:   ;print signon message and go to ccp
208                             ;       (note: mds boot initialized iobyte at 0003h)
209     16b3 310001                     lxi     sp, buff+80h
210     16b6 219c16                     lxi     h, signon
211     16b9 cdd317                     call    prmsg           ;print message
212     16bc af                         xra     a               ;clear accumulator
213     16bd 320400                     sta     cdisk           ;set initially to disk a
214     16c0 c30f17                     jmp     gocpm           ;go to cp/m
215                             ;
216                             ;
217                             wboot:; loader on track 0, sector 1, which will be skipped for warm
218                             ;       read cp/m from disk—assuming there is a 128 byte cold start
219                             ;       start
220                             ;
221     16c3 318000                     lxi     sp, buff        ;using dma—thus 80 thru ff available for stack
222                             ;
223     16c6 0e0a                       mvi     c, retry        ;max retries
224     16c8 c5                         push    b
225                             wboot0: ;enter here on error retries
226     16c9 010000                     lxi     b, cpmb         ;set dma address to start of disk system
227     16cc cdbb17                     call    setdma
228     16cf 0e00                       mvi     c, 0            ;boot from drive 0
229     16d1 cd7d17                     call    seldsk
230     16d4 0e00                       mvi     c, 0
231     16d6 cda717                     call    settrk          ;start with track 0
232     16d9 0e02                       mvi     c, 2            ;start reading sector 2
233     16db cdac17                     call    setsec
```

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
234                         ;
235                         ;       read sectors, count nsects to zero
236     16de c1                     pop     b           ;10-error count
237     16df 062c                   mvi     b, nsects
238                         rdsec:  ;read next sector
239     16e1 c5                     push    b           ;save sector count
240     16e2 cdc117                 call    read
241     16e5 c24917                 jnz     booterr     ;retry if errors occur
242     16e8 2a6c18                 lhld    iod         ;increment dma address
243     16eb 118000                 lxi     d, 128      ;sector size
244     16ee 19                     dad     d           ;incremented dma address in hl
245     16ef 44                     mov     b, h
246     16f0 4d                     mov     c, l        ;ready for call to set dma
247     16f1 cdbb17                 call    setdma
248     16f4 3a6b18                 lda     ios         ;sector number just read
249     16f7 fe1a                   cpi     26          ;read last sector?
250     16f9 da0517                 jc      rd1
251                         ;       must be sector 26, zero and go to next track
252     16fc 3a6a18                 lda     iot         ;get track to register a
253     16ff 3c                     inr     a
254     1700 4f                     mov     c, a        ;ready for call
255     1701 cda717                 call    settrk
256     1704 af                     xra     a           ;clear sector number
257     1705 3c             rd1:    inr     a           ;to next sector
258     1706 4f                     mov     c, a        ;ready for call
259     1707 cdac17                 call    setsec
260     170a c1                     pop     b           ;recall sector count
261     170b 05                     dcr     b           ;done?
262     170c c2e116                 jnz     rdsec
263                         ;
264                         ;       done with the load, reset default buffer address
265                         gocpm:  ;(enter here from cold start boot)
266                         ;       enable rst0 and rst7
```

```
267     170f f3                 di
268     1710 3e12               mvi     a, 12h          ;initialize command
269     1712 d3fd               out     revrt
270     1714 af                 xra     a
271     1715 d3fc               out     intc            ;cleared
272     1717 3e7e               mvi     a, inte         ;rst0 and rst7 bits on
273     1719 d3fc               out     intc
274     171b af                 xra     a
275     171c d3f3               out     icon            ;interrupt control
276                     ;
277                     ;       set default buffer address to 80h
278     171e 018000             lxi     b, buff
279     1721 cdbb17             call    setdma
280                     ;
281                     ;       reset monitor entry points
282     1724 3ec3               mvi     a, jmp
283     1726 320000             sta     0
284     1729 210316             lxi     h, wboote
285     172c 220100             shld    1               ;jump wboot at location 00
286     172f 320500             sta     5
287     1732 210608             lxi     h, bdos
288     1735 220600             shld    6               ;jmp bdos at location 5
289                             if      not test
290     1738 323800             sta     7*8             ;jmp to mon80 (may have changed by ddt)
291     173b 2100f8             lxi     h, mon80
292     173e 223900             shld    7*8+1
293                             endif
294                     ;       leave iobyte set
295                     ;       previously selected disk was b, send parameter to cpm
296     1741 3a0400             lda     cdisk           ;last logged disk number
297     1744 4f                 mov     c, a            ;send to ccp to log it in
298     1745 fb                 ei
299     1746 c30000             jmp     cpmb
300                     ;
```

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
301                     ;       error condition occurred, print message and retry
302                     booterr:
303     1749 c1                 pop     b               ;recall counts
304     174a 0d                 dcr     c
305     174b ca5217             jz      booter0
306                     ;       try again
307     174e c5                 push    b
308     174f c3c916             jmp     wboot0
309                     ;
310                     booter0:
311                     ;       otherwise too many retries
312     1752 215b17             lxi     h, bootmsg
313     1755 cdd317             call    prmsg
314     1758 c30fff             jmp     rmon80          ;mds hardware monitor
315                     ;
316                     bootmsg:
317     175b 3f626f6f74         db      '?boot', 0
318                     ;
319                     ;
320                     const:  console status to reg-a
321                     ;       (exactly the same as mds call)
322     1761 c312f8             jmp     csts
323                     ;
324                     conin:  ;console character to reg-a
325     1764 cd03f8             call    ci
326     1767 e67f               ani     7fh             ;remove parity bit
327     1769 c9                 ret
328                     ;
329                     conout: ;console character from c to console out
330     176a c309f8             jmp     co
331                     ;
332                     list:   ;list device out
333                     ;       (exactly the same as mds call)
334     176d c30ff8             jmp     lo
```

```
335                             ;
336                             listst:
337                                     ;return list status
338     1770 af                         xra     a
339     1771 c9                         ret                     ;always not ready
340                             ;
341                             punch:  ;punch device out
342                             ;       (exactly the same as mds call)
343     1772 c30cf8                     jmp     po
344                             ;
345                             reader: ;reader character in to reg-a
346                             ;       (exactly the same as mds call)
347     1775 c306f8                     jmp     ri
348                             ;
349                             home:   ;move to home position
350                             ;       treat as track 00 seek
351     1778 0e00                       mvi     c, 0
352     177a c3a717                     jmp     settrk
353                             ;
354                             seldsk: ;select disk given by register c
355     177d 210000                     lxi     h, 0000h        ;return 0000 if error
356     1780 79                         mov     a, c
357     1781 fe04                       cpi     ndisks          ;too large?
358     1783 d0                         rnc                     ;leave hl = 0000
359                             ;
360     1784 e602                       ani     10b             ;00 00 for drive 0, 1 and 10 10 for drive 2, 3
361     1786 326618                     sta     dbank           ;to select drive bank
362     1789 79                         mov     a, c            ;00, 01, 10, 11
363     178a e601                       ani     1b              ;mds has 0, 1 at 78, 2, 3 at 88
364     178c b7                         ora     a               ;result 00?
365     178d ca9217                     jz      setdrive
366     1790 3e30                       mvi     a, 00110000b        ;selects drive 1 in bank
367                             setdrive:
368     1792 47                         mov     b, a            ;save the function
```

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
369     1793 216818             lxi     h, iof      ;io function
370     1796 7e                 mov     a, m
371     1797 e6cf               ani     11001111b   ;mask out disk number
372     1799 b0                 ora     b           ;mask in new disk number
373     179a 77                 mov     m, a        ;save it in iopb
374     179b 69                 mov     l, c
375     179c 2600               mvi     h, 0        ;hl=disk number
376     179e 29                 dad     h           ;*2
377     179f 29                 dad     h           ;*4
378     17a0 29                 dad     h           ;*8
379     17a1 29                 dad     h           ;*16
380     17a2 113316             lxi     d, dpbase
381     17a5 19                 dad     d           ;hl=disk header table address
382     17a6 c9                 ret
383                     ;
384                     ;
385                     settrk: ;set track address given by c
386     17a7 216a18             lxi     h, iot
387     17aa 71                 mov     m, c
388     17ab c9                 ret
389                     ;
390                     setsec: ;set sector number given by c
391     17ac 216b18             lxi     h, ios
392     17af 71                 mov     m, c
393     17b0 c9                 ret
394                     sectran:
395                             ;translate sector bc using table at de
396     17b1 0600               mvi     b, 0        ;double precision sector number in bc
397     17b3 eb                 xchg                ;translate table address to hl
398     17b4 09                 dad     b           ;translate (sector) address
399     17b5 7e                 mov     a, m        ;translated sector number to a
400     17b6 326b18             sta     ios
401     17b9 6f                 mov     l, a        ;return sector number in l
402     17ba c9                 ret
```

```
403                     ;
404                     setdma:  ;set dma address given by regs b, c
405     17bb 69                  mov     l, c
406     17bc 60                  mov     h, b
407     17bd 226c18              shld    iod
408     17c0 c9                  ret
409                     ;
410                     read:    ;read next disk record (assuming disk/trk/ sec/dma set)
411     17c1 0e04                mvi     c, readf        ;set to read function
412     17c3 cde017              call    setfunc
413     17c6 cdf017              call    waitio          ;perform read function
414     17c9 c9                  ret                     ;may have error set in reg-a
415                     ;
416                     ;
417                     write:   ;disk write function
418     17ca 0e06                mvi     c, writf
419     17cc cde017              call    setfunc         ;set to write function
420     17cf cdf017              call    waitio
421     17d2 c9                  ret                     ;may have error set
422                     ;
423                     ;
424                     ;        utility subroutines
425                     prmsg:   ;print message at h, l to 0
426     17d3 7e                  mov     a, m
427     17d4 b7                  ora     a               zero?
428     17d5 c8                  rz
429                     ;        more to print
430     17d6 e5                  push    h
431     17d7 4f                  mov     c, a
432     17d8 cd6a17              call    conout
433     17db e1                  pop     h
434     17dc 23                  inx     h
435     17dd c3d317              jmp     prmsg
436                     ;
```

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
437                     setfunc:
438                     ;       set function for next i/o (command in reg-c)
439     17e0 216818             lxi     h, iof          ;io function address
440     17e3 7e                 mov     a, m            ;get it to accumulator for masking
441     17e4 e6f8               ani     11111000b       ;remove previous command
442     17e6 b1                 ora     c               ;set to new command
443     17e7 77                 mov     m, a            ;replaced in iopb
444                     ;       the mds-800 controller requires disk bank bit in sector byte
445                     ;       mask the bit from the current i/o function
446     17e8 e620               ani     00100000b               ;mask the disk select bit
447     17ea 216b18             lxi     h, ios                  ;address the sector select byte
448     17ed b6                 ora     m                       ;select proper disk bank
449     17ee 77                 mov     m, a                    ;set disk select bit on/off
450     17ef c9                 ret
451                     ;
452                     waitio:
453     17f0 0e0a               mvi     c, retry        ;max retries before perm error
454                     rewait:
455                     ;       start the i/o function and wait for completion
456     17f2 cd3f18             call    intype          ;in rtype
457     17f5 cd4c18             call    inbyte          ;clears the controller
458                     ;
459     17f8 3a6618             lda     dbank                   ;set bank flags
460     17fb b7                 ora     a                       ;zero if drive 0, 1 and nz if 2, 3
461     17fc 3e67               mvi     a, iopb and offh        ;low address for iopb
462     17fe 0618               mvi     b, iopb shr 8           ;high address for iopb
463     1800 c20b18             jnz     iodr1           ;drive bank 1?
464     1803 d379               out     ilow                    ;low address to controller
465     1805 78                 mov     a, b
466     1806 d37a               out     ihigh           ;high address
467     1808 c31018             jmp     waito                   ;to wait for complete
468                     ;
469                     iodr1:  ;drive bank 1
470     180b d389               out     ilow+10h                ;88 for drive bank 10
```

```
471     180d 78                 mov     a, b
472     180e d38a               out     ihigh+10h
473                     ;
474     1810 cd5918     waito:  call    instat          ;wait for completion
475     1813 e604               ani     iordy           ;ready?
476     1815 ca1018             jz      waito
477                     ;
478                     ;       check io completion ok
479     1818 cd3f18             call    intype          ;must be io complete (00) unlinked
480                     ;       00 unlinked i/o complete, 01 linked i/o complete (not used)
481                     ;       io disk status changed  11 (not used)
482     181b fe02               cpi     10b             ;ready status change?
483     181d ca3218             jz      wready
484                     ;
485                     ;       must be 00 in the accumulator
486     1820 b7                 ora     a
487     1821 c23818             jnz     werror          ;some other condition, retry
488                     ;
489                     ;       check i/o error bits
490     1824 cd4c18             call    inbyte
491     1827 17                 ral
492     1828 da3218             jc      wready          ;unit not ready
493     182b 1f                 rar
494     182c e6fe               ani     11111110b       ;any other errors? (deleted data ok)
495     182e c23818             jnz     werror
496                     ;
497                     ;       read or write is ok, accumulator contains zero
498     1831 c9                 ret
499                     ;
500                     wready: ;not ready, treat as error for now
501     1832 cd4c18             call    inbyte          ;clear result byte
502     1835 c33818             jmp     trycount
503                     ;
504                     werror: ;return hardware malfunction (crc, track, seek, etc.)
```

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
505                     ;       the mds controller has returned a bit in each position
506                     ;       of the accumulator, corresponding to the conditions:
507                     ;       0       —deleted data (accepted as ok above)
508                     ;       1       —crc error
509                     ;       2       —seek error
510                     ;       3       —address error (hardware malfunction)
511                     ;       4       —data over/under flow (hardware malfunction)
512                     ;       5       —write protect (treated as not ready)
513                     ;       6       —write error (hardware malfunction)
514                     ;       j       —not ready
515                     ;       (accumulator bits are numbered 7 6 5 4 3 2 1 0)
516                     ;
517                     ;       it may be useful to filter out the various conditions,
518                     ;       but we will get a permanent error message if it is not
519                     ;       recoverable.  in any case, the not ready condition is
520                     ;       treated as a separated condition for later improvement
521                     trycount:
522                     ;       register c contains retry count, decrement 'til zero
523     1838 0d                 dcr     c
524     1839 c2f217             jnz     rewait          ;for another try
525                     ;
526                     ;       cannot recover from error
527     183c 3e01               mvi     a, 1            ;error code
528     183e c9                 ret
529                     ;
530                     ;       intype, inbyte, instat read drive bank 00 or 10
531     183f 3a6618     intype: lda     dbank
532     1842 b7                 ora     a
533     1843 c24918             jnz     intyp1          ;skip to bank 10
534     1846 db79               in      rtype
535     1848 c9                 ret
536     1849 db89       intyp1: in      rtype+10h               ;78 for 0, 1  88 for 2, 3
537     184b c9                 ret
```

```
538                             ;
539     184c 3a6618     inbyte: lda     dbank
540     184f b7                 ora     a
541     1850 c25618             jnz     inbyt1
542     1853 db7b               in      rbyte
543     1855 c9                 ret
544     1856 db8b       inbyt1: in      rbyte+10h
545     1858 c9                 ret
546                             ;
547     1859 3a6618     instat: lda     dbank
548     185c b7                 ora     a
549     185d c26318             jnz     insta1
550     1860 db78               in      dstat
551     1862 c9                 ret
552     1863 db88       insta1: in      dstat+10h
553     1865 c9                 ret
554                             ;
555                             ;
556                             ;
557                             ;       data areas (must be in ram)
558     1866 00         dbank:  db      0               ;disk bank 00 if drive 0, 1
559                                                     ;               10 if drive 2, 3
560                             iopb:   ;io parameter block
561     1867 80                 db      80h             ;normal i/o operation
562     1868 04         iof:    db      readf           ;io function, initial read
563     1869 01         ion:    db      1               ;number of sectors to read
564     186a 02         iot:    db      offset          ;track number
565     186b 01         ios:    db      1               ;sector number
566     186c 8000       iod:    dw      buff            ;io address
567                             ;
568                             ;
569                             ;       define ram areas for bdos operation
570                             endef
```

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
571     186e+=          begdat  equ     $
572     186e+           dirbuf: ds      128     ;directory access buffer
573     18ee+           alv0:   ds      31
574     190d+           csv0:   ds      16
575     191d+           alv1:   ds      31
576     193c+           csv1:   ds      16
577     194c+           alv2:   ds      31
578     196b+           csv2:   ds      16
579     197b+           alv3:   ds      31
580     199a+           csv3:   ds      16
581     19aa+=          enddat  equ     $
582     013c+=          datsiz  equ     $-begdat
583     19aa                    end
```

```
als1            001f    141#
als2            001f    146#
als3            001f    151#
alv0            18ee     87     573#
alv1            191d     91     575#
alv2            194c     95     577#
alv3            197b     99     579#
base            0078    180#    181     182     183     185     186
bdos            0806     29#    287
begdat          186e    571#    582
bias            0000     19#     22#
boot            16b3     63     207#
booter0         1752    305     310#
booterr         1749    241     302#
bootmsg         175b    312     316#
buff            0080     34#    209     221     278     566
cdisk           0004     33#    213     296
ci              f803    172#    325
co              f809    174#    330
```

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

conin	1764	66	324#				
conout	176a	67	329#	432			
const	1761	65	320#				
cpmb	0000	28#	29	30	226	299	
cpml	1600	30#	31				
cr	000d	192#	196	205			
css1	0010	142#					
css2	0010	147#					
css3	0010	152#					
csts	f812	177#	322				
csv0	190d	87	574#				
csv1	193c	91	576#				
csv2	196b	95	578#				
csv3	199a	99	580#				
datsiz	013c	582#					
dbank	1866	361	459	531	539	547	558#
dirbuf	186e	86	90	94	98	572#	
dpb0	1673	86	101#	140	145	150	
dpb1	1673	90	140#				
dpb2	1673	94	145#				
dpb3	1673	98	150#				
dpbase	1633	83#	380				
dpe0	1633	84#					
dpe1	1643	88#					
dpe2	1653	92#					
dpe3	1663	96#					
dstat	0078	181#	550	552			
enddat	19aa	581#					
false	0000	15#	16				
gocpm	170f	214	265#				
home	1778	71	349#				
icon	00f3	166#	275				
ihigh	007a	186#	466	472			

ilow	0079	185#	464	470		
inbyt1	1856	541	544#			
inbyte	184c	457	490	501	539#	
insta1	1863	549	552#			
instat	1859	474	547#			
intc	00fc	165#	271	273		
inte	007e	167#	272			
intyp1	1849	533	536#			
intype	183f	456	479	531#		
iod	186c	242	407	566#		
iodr1	180b	463	469#			
iof	1868	369	439	562#		
ion	1869	563#				
iopb	1867	461	462	560#		
iordy	0004	191#	475			
ios	186b	248	391	400	447	565#
iot	186a	252	386	564#		
lf	000a	193#	196	196	205	
list	176d	68	332#			
listst	1770	78	336#			
lo	f80f	176#	334			
mon80	f800	170#	291			
nsects	002c	31#	237			
offset	0002	32#	100	564		
patch	1600	25#	27	28		
po	f80c	175#	343			
prmsg	17d3	211	313	425#	435	
punch	1772	69	341#			
rbyte	007b	183#	542	544		
rd1	1705	250	257#			
rdsec	16e1	238#	262			
read	17c1	76	240	410#		
reader	1775	70	345#			
readf	0004	188#	411	562		

recal	0003	190#					
retry	000a	35#	223	453			
revrt	00fd	164#	269				
rewait	17f2	454#	524				
ri	f806	173#	347				
rmon80	ff0f	171#	314				
rtype	0079	182#	534	536			
sectran	17b1	79	394#				
seldsk	177d	72	229	354#			
setdma	17bb	75	227	247	279	404#	
setdrive	1792	365	367#				
setfunc	17e0	412	419	437#			
setsec	17ac	74	233	259	390#		
settrk	17a7	73	231	255	352	385#	
signon	169c	195#	210				
test	0000	16#	18	21	197	200	289
true	ffff	14#	15				
trycount	1838	502	521#				
vers	0016	6#	204	204			
waito	1810	467	474#	476			
waitio	17f0	413	420	452#			
wboot	16c3	64	217#				
wboot0	16c9	225#	308				
wboote	1603	64#	284				
werror	1838	487	495	504#			
wready	1832	483	492	500#			
write	17ca	77	417#				
writf	0006	189#	418				
xlt0	1682	84	112#	143	148	153	
xlt1	1682	88	143#				
xlt2	1682	92	148#				
xlt3	1682	96	153#				

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

Appendix B: A Skeletal CBIOS

```
                ;       skeletal cbios for first level of cp/m 2.0 alteration
                ;
0014 =          msize   equ     20              ;cp/m version memory size in kilobytes
                ;
                ;       "bias" is address offset from 3400h for memory systems
                ;       than 16k (referred to as "b" throughout the text)
                ;
0000 =          bias    equ     (msize-20)*1024
3400 =          ccp     equ     3400h+bias      ;base of ccp
3c06 =          bdos    equ     ccp+806h        ;base of bdos
4a00 =          bios    equ     ccp+1600h       ;base of bios
0004 =          cdisk   equ     0004h           ;current disk number  0=a, . . . , 15=p
0003 =          iobyte  equ     0003h           ;intel i/o byte
                ;
4a00                    org     bios            ;origin of this program
002c =          nsects  equ     ($-ccp)/128     ;warm start sector count
                ;
                ;       jump vector for individual subroutines
4a00 c39c4a             jmp     boot            ;cold start
4a03 c3a64a     wboote: jmp     wboot           ;warm start
4a06 c3114b             jmp     const           ;console status
4a09 c3244b             jmp     conin           ;console character in
4a0c c3374b             jmp     conout          ;console character out
4a0f c3494b             jmp     list            ;list character out
4a12 c34d4b             jmp     punch           ;punch character out
4a15 c34f4b             jmp     reader          ;reader character out
4a18 c3544b             jmp     home            ;move head to home position
```

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
28      4a1b c35a4b                     jmp     seldsk      ;select disk
29      4a1e c37d4b                     jmp     settrk      ;set track number
30      4a21 c3924b                     jmp     setsec      ;set sector number
31      4a24 c3ad4b                     jmp     setdma      ;set dma address
32      4a27 c3c34b                     jmp     read        ;read disk
33      4a2a c3d64b                     jmp     write       ;write disk
34      4a2d c34b4b                     jmp     listst      ;return list status
35      4a30 c3a74b                     jmp     sectran     ;sector translate
36                              ;
37                              ;       fixed data tables for four-drive standard
38                              ;       ibm-compatible 8" disks
39                              ;       disk parameter header for disk 00
40      4a33 734a0000           dpbase: dw      trans, 0000h
41      4a37 00000000                   dw      0000h, 0000h
42      4a3b f04c8d4a                   dw      dirbf, dpblk
43      4a3f ec4d704d                   dw      chk00, all00
44                              ;       disk parameter header for disk 01
45      4a43 734a0000                   dw      trans, 0000h
46      4a47 00000000                   dw      0000h, 0000h
47      4a4b f04c8d4a                   dw      dirbf, dpblk
48      4a4f fc4d8f4d                   dw      chk01, all01
49                              ;       disk parameter header for disk 02
50      4a53 734a0000                   dw      trans, 0000h
51      4a57 00000000                   dw      0000h, 0000h
52      4a5b f04c8d4a                   dw      dirbf, dpblk
53      4a5f 0c4eae4d                   dw      chk02, all02
54                              ;       disk parameter header for disk 03
55      4a63 734a0000                   dw      trans, 0000h
56      4a67 00000000                   dw      0000h, 0000h
57      4a6b f04c8d4a                   dw      dirbf, dpblk
58      4a6f 1c4ecd4d                   dw      chk03, all03
59                              ;
60                              ;       sector translate vector
```

```
61    4a73 01070d13   trans:  db      1, 7, 13, 19    ;sectors 1, 2, 3, 4
62    4a77 19050b11           db      25, 5, 11, 17   ;sectors 5, 6, 7, 8
63    4a7b 1703090f           db      23, 3, 9, 15    ;sectors 9, 10, 11, 12
64    4a7f 1502080e           db      21, 2, 8, 14    ;sectors 13, 14, 15, 16
65    4a83 141a060c           db      20, 26, 6, 12   ;sectors 17, 18, 19, 20
66    4a87 1218040a           db      18, 24, 4, 10   ;sectors 21, 22, 23, 24
67    4a8b 1016               db      16, 22          ;sectors 25, 26
68                    ;
69                    dpblk:  ;disk parameter block, common to all disks
70    4a8d 1a00               dw      26              ;sectors per track
71    4a8f 03                 db      3               ;block shift factor
72    4a90 07                 db      7               ;block mask
73    4a91 00                 db      0               ;null mask
74    4a92 f200               dw      242             ;disk size-1
75    4a94 3f00               dw      63              ;directory max
76    4a96 c0                 db      192             ;alloc 0
77    4a97 00                 db      0               ;alloc 1
78    4a98 1000               dw      16              ;check size
79    4a9a 0200               dw      2               ;track offset
80                    ;
81                    ;       end of fixed tables
82                    ;
83                    ;       individual subroutines to perform each function
84                    boot:   ;simplest case is to just perform parameter initialization
85    4a9c af                 xra     a               ;zero in the accum
86    4a9d 320300             sta     iobyte          ;clear the iobyte
87    4aa0 320400             sta     cdisk           ;select disk zero
88    4aa3 c3ef4a             jmp     gocpm           ;initialize and go to cp/m
89                    ;
90                    wboot:  ;simplest case is to read the disk until all sectors loaded
91    4aa6 318000             lxi     sp, 80h         ;use space below buffer for stack
92    4aa9 0e00               mvi     c, 0            ;select disk 0
93    4aab cd5a4b             call    seldsk
94    4aae cd544b             call    home            ;go to track 00
```

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
                        ;
4ab1 062c               mvi     b, nsects       ;b counts # of sectors to load
4ab3 0e00               mvi     c, 0            ;c has the current track number
4ab5 1602               mvi     d, 2            ;d has the next sector to read
                        ;       note that we begin by reading track 0, sector 2 since sector 1
                        ;       contains the cold start loader, which is skipped in a warm start
4ab7 210034             lxi     h, ccp          ;base of cp/m (initial load point)
                load1:  ;load one more sector
4aba c5                 push    b               ;save sector count, current track
4abb d5                 push    d               ;save next sector to read
4abc e5                 push    h               ;save dma address
4abd 4a                 mov     c, d            ;get sector address to register c
4abe cd924b             call    setsec          ;set sector address from register c
4ac1 c1                 pop     b               ;recall dma address to b, c
4ac2 c5                 push    b               ;replace on stack for later recall
4ac3 cdad4b             call    setdma          ;set dma address from b, c
                        ;
                        ;       drive set to 0, track set, sector set, dma address set
4ac6 cdc34b             call    read
4ac9 fe00               cpi     00h             ;any errors?
4acb c2a64a             jnz     wboot           ;retry the entire boot if an error occurs
                        ;
                        ;       no error, move to next sector
4ace e1                 pop     h               ;recall dma address
4acf 118000             lxi     d, 128          ;dma=dma+128
4ad2 19                 dad     d               ;new dma address is in h, l
4ad3 d1                 pop     d               ;recall sector address
4ad4 c1                 pop     b               ;recall number of sectors remaining, and current trk
4ad5 05                 dcr     b               ;sectors=sectors-1
4ad6 caef4a             jz      gocpm           ;transfer to cp/m if all have been loaded
                        ;
                        ;       more sectors remain to load, check for track change
4ad9 14                 inr     d
```

```
128  4ada 7a                mov   a, d          ;sector=27?, if so, change tracks
129  4adb fe1b              cpi   27
130  4add daba4a            jc    load1         ;carry generated if sector<27
131                         ;
132                         ;     end of current track, go to next track
133  4ae0 1601              mvi   d, 1          ;begin with first sector of next track
134  4ae2 0c                inr   c             ;track=track+1
135                         ;
136                         ;     save register state, and change tracks
137  4ae3 c5                push  b
138  4ae4 d5                push  d
139  4ae5 e5                push  h
140  4ae6 cd7d4b            call  settrk        ;track address set from register c
141  4ae9 e1                pop   h
142  4aea d1                pop   d
143  4aeb c1                pop   b
144  4aec c3ba4a            jmp   load1         ;for another sector
145                         ;
146                         ;     end of load operation, set parameters and go to cp/m
147                         gocpm:
148  4aef 3ec3              mvi   a, 0c3h       ;c3 is a jmp instruction
149  4af1 320000            sta   0             ;for jmp to wboot
150  4af4 21034a            lxi   h, wboote     ;wboot entry point
151  4af7 220100            shld  1             ;set address field for jmp at 0
152                         ;
153  4afa 320500            sta   5             ;for jmp to bdos
154  4afd 21063c            lxi   h, bdos       ;bdos entry point
155  4b00 220600            shld  6             ;address field of jump at 5 to bdos
156                         ;
157  4b03 018000            lxi   b, 80h        ;default dma address is 80h
158  4b06 cdad4b            call  setdma
159                         ;
160  4b09 fb                ei                  ;enable the interrupt system
```

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
161     4b0a 3a0400             lda     cdisk       ;get current disk number
162     4b0d 4f                 mov     c, a        ;send to the ccp
163     4b0e c30034             jmp     ccp         ;go to cp/m for further processing
164                     ;
165                     ;
166                     ;       simple i/o handlers (must be filled in by user)
167                     ;       in each case, the entry point is provided, with space reserved
168                     ;       to insert your own code
169                     ;
170                     const:  ;console status, return 0ffh if character ready, 00h if not
171     4b11                    ds      10h         ;space for status subroutine
172     4b21 3e00               mvi     a, 00h
173     4b23 c9                 ret
174                     ;
175                     conin:  ;console character into register a
176     4b24                    ds      10h         ;space for input routine
177     4b34 e67f               ani     7fh         ;strip parity bit
178     4b36 c9                 ret
179                     ;
180                     conout: ;console character output from register c
181     4b37 79                 mov     a, c        ;get to accumulator
182     4b38                    ds      10h         ;space for output  routine
183     4b48 c9                 ret
184                     ;
185                     list:   ;list character from register c
186     4b49 79                 mov     a, c        ;character to register a
187     4b4a c9                 ret                 ;null subroutine
188                     ;
189                     listst: ;return list status (0 if not ready, 1 if ready)
190     4b4b af                 xra     a           ;0 is always ok to return
191     4b4c c9                 ret
192                     ;
193                     punch:  ;punch character from register c
```

```
194     4b4d 79                 mov     a, c            ;character to register a
195     4b4e c9                 ret                     ;null subroutine
196                     ;
197                     ;
198                     reader: ;read character into register a from reader device
199     4b4f 3e1a               mvi     a, 1ah          ;enter end of file for  now (replace later)
200     4b51 e67f               ani     7fh             ;remember to strip parity bit
201     4b53 c9                 ret
202                     ;
203                     ;
204                     ;       i/o drivers for the disk follow
205                     ;       for now, we will simply store the parameters away for use
206                     ;       in the read and write subroutines
207                     ;
208                     home:   ;move to the track 00 position of current  drive
209                     ;       translate this call into a settrk call with parameter 00
210     4b54 0e00               mvi     c, 0            ;select track 0
211     4b56 cd7d4b             call    settrk
212     4b59 c9                 ret                     ;we will move to 00 on first read/write
213                     ;
214                     seldsk: ;select disk given by register c
215     4b5a 210000             lxi     h, 0000h        ;error return code
216     4b5d 79                 mov     a,c
217     4b5e 32ef4c             sta     diskno
218     4b61 fe04               cpi     4               ;must be between 0 and  3
219     4b63 d0                 rnc                     ;no carry if 4, 5, . . .
220                     ;       disk number is in the proper range
221     4b64                    ds      10              ;space for disk select
222                     ;       compute proper disk parameter header address
223     4b6e 3aef4c             lda     diskno
224     4b71 6f                 mov     l, a            ;l=disk number 0, 1, 2, 3
225     4b72 2600               mvi     h, 0            ;high order zero
226     4b74 29                 dad     h               ;*2
```

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
227     4b75 29                 dad     h           ;*4
228     4b76 29                 dad     h           ;*8
229     4b77 29                 dad     h           ;*16 (size of each header)
230     4b78 11334a             lxi     d, dpbase
231     4b7b 19                 dad     0           ;hl=.dpbase(diskno*16)
232     4b7c c9                 ret
233                     ;
234                     settrk: ;set track given by register c
235     4b7d 79                 mov     a, c
236     4b7e 32e94c             sta     track
237     4b81                    ds      10h         ;space for track select
238     4b91 c9                 ret
239                     ;
240                     setsec: ;set sector given by register c
241     4b92 79                 mov     a, c
242     4b93 32eb4c             sta     sector
243     4b96                    ds      10h         ;space for sector select
244     4ba6 c9                 ret
245                     ;
246                     sectran:
247                             ;translate the sector given by bc using the
248                             ;translate table given by de
249     4ba7 eb                 xchg                ;hl=.trans
250     4ba8 09                 dad     b           ;hl=.trans(sector)
251     4ba9 6e                 mov     l, m        ;l = trans(sector)
252     4baa 2600               mvi     h, 0        ;hl = trans(sector)
253     4bac c9                 ret                 ;with value in hl
254                     ;
255                     setdma: ;set dma address given by registers b and c
256     4bad 69                 mov     l, c        ;low order address
257     4bae 60                 mov     h, b        ;high order address
258     4baf 22ed4c             shld    dmaad       ;save the address
259     4bb2                    ds      10h         ;space for setting the dma address
```

```
260     4bc2 c9                         ret
261                             ;
262                             read:   ;perform read operation (usually this is similar to write
263                             ;       so we will allow space to set up read command, then use
264                             ;       common code in write)
265     4bc3                            ds      10h             ;set up read command
266     4bd3 c3e64b                     jmp     waitio          ;to perform the actual i/o
267                             ;
268                             write:  ;perform a write operation
269     4bd6                            ds      10h             ;set up write command
270                             ;
271                             waitio: ;enter here from read and write to perform the actual i/o
272                             ;       operation. return a 00h in register a if the operation completes
273                             ;       properly, and 01h if an error occurs during the read or write
274                             ;
275                             ;       in this case, we have saved the disk number in 'diskno' (0, 1)
276                             ;                               the track number in 'track' (0-76)
277                             ;                               the sector number in 'sector' (1-26)
278                             ;                               the dma address in 'dmaad' (0-65535)
279     4be6                            ds      256             ;space reserved for i/o drivers
280     4ce6 3e01                       mvi     a, 1            ;error condition
281     4ce8 c9                         ret                     ;replaced when filled- in
282                             ;
283                             ;       the remainder of the cbios is reserved uninitialized
284                             ;       data area, and does not need to be a part of the
285                             ;       system memory image (the space must be available,
286                             ;       however, between "begdat" and "enddat").
287                             ;
288     4ce9                    track:  ds      2               ;two bytes for expansion
289     4ceb                    sector: ds      2               ;two bytes for expansion
290     4ced                    dmaad:  ds      2               ;direct memory address
```

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
291     4cef            diskno:  ds      1           ;disk number 0-15
292                     ;
293                     ;        scratch ram area for bdos use
294     4cf0 =          begdat   equ     $           ;beginning of data area
295     4cf0            dirbf:   ds      128         ;scratch directory area
296     4d70            all00:   ds      31          ;allocation vector 0
297     4d8f            all01:   ds      31          ;allocation vector 1
298     4dae            all02:   ds      31          ;allocation vector 2
299     4dcd            all03:   ds      31          ;allocation vector 3
300     4dec            chk00:   ds      16          ;check vector 0
301     4dfc            chk01:   ds      16          ;check vector 1
302     4e0c            chk02:   ds      16          ;check vector 2
303     4e1c            chk03:   ds      16          ;check vector 3
304                     ;
305     4e2c            enddat   equ     $           ;end of data area
306     013c =          datsiz   equ     $-begdat;   ;size of data area
307     4e2c                     end
```

```
all00           4d70    43      296#
all01           4d8f    48      297#
all02           4dae    53      298#
all03           4dcd    58      299#
bdos            3c06    10#     154
begdat          4cf0    294#    306
bias            0000    8#      9
bios            4a00    11#     15
boot            4a9c    19      84#
ccp             3400    9#      10      11      16      101     163
cdisk           0004    12#     87      161
chk00           4dec    43      300#
chk01           4dfc    48      301#
chk02           4e0c    53      302#
chk03           4e1c    58      303#
```

conin	4b24	22	175#			
conout	4b37	23	180#			
const	4b11	21	170#			
datsiz	013c	306#				
dirbf	4cf0	42	47	52	57	295#
diskno	4cef	217	223	291#		
dmaad	4ced	258	290#			
dpbase	4a33	40#	230			
dpblk	4a8d	42	47	52	57	69#
enddat	4e2c	305#				
gocpm	4aef	88	124	147#		
home	4b54	27	94	208#		
iobyte	0003	13#	86			
list	4b49	24	185#			
liststt	4b4b	34	189#			
load1	4aba	102#	130	144		
msize	0014	3#	8			
nsects	002c	16#	96			
punch	4b4d	25	193#			
read	4bc3	32	113	262#		
reader	4b4f	26	198#			
sector	4ceb	242	289#			
sectran	4ba7	35	246#			
seldsk	4b5a	28	93	214#		
setdma	4bad	31	110	158	255#	
setsec	4b92	30	107	240#		
settrk	4b7d	29	140	211	234#	
track	4ce9	236	288#			
trans	4a73	40	45	50	55	61#
waitio	4be6	266	271#			
wboot	4aa6	20	90#	115		
wboote	4a03	20#	150			
write	4bd6	33	268#			

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

Appendix C: A Skeletal GETSYS/PUTSYS Program

```
                        ;       combined getsys and putsys programs from
                        ;       Sec 6.4
                        ;       Start the programs at the base of the TPA

0100                            org  0100h

0014 =                  msize   equ  20             ; size of cp/m in Kbytes

                        ; "bias" is the amount to add to addresses for > 20k
                        ;       (referred to as "b" throughout the text)

0000 =                  bias    equ  (msize-20)*1024
3400 =                  ccp     equ  3400h+bias
3c00 =                  bdos    equ  ccp+0800h
4a00 =                  bios    equ  ccp+1600h

                        ;       getsys programs tracks 0 and 1 to memory at
                        ;       3880h + bias

                        ;       register            usage
                        ;       a                   (scratch register)
                        ;       b                   track count (0...76)
                        ;       c                   sector count (1...26)
                        ;       d,e                 (scratch register pair)
                        ;       h,l                 load address
                        ;       sp                  set to track address

                        gstart:                     ; start of getsys
0100 318033                     lxi  sp,ccp-0080h   ; convenient place
0103 218033                     lxi  h,ccp-0080h    ; set initial load
0106 0600                       mvi  b,0            ; start with track
                        rd$trk:                     ; read next track
```

```
0108 0e01                       mvi  c,1            ; each track start
                        rd$sec:
010a cd0003                     call read$sec       ; get the next sector
010d 118000                     lxi  d,128          ; offset by one sector
0110 19                         dad  d              ;  (hl=hl+128)
0111 0c                         inr  c              ; next sector
0112 79                         mov a,c             ; fetch sector number
0113 felb                       cpi  27             ;  and see if last
0115 da0a01                     jc   rdsec          ; <, do one more

                        ; arrive here at end of track, move to next track

0118 04                         inr  b              ; track = track+1
0119 78                         mov a,b             ; check for last
011a fe02                       cpi  2              ; track = 2 ?
011c da0801                     jc   rd$trk         ; <, do another

                        ; arrive here at end of load, halt for lack of anything
                        ; better

011f fb                         ei
0120 76                         hlt
                        ;       putsys program, places memory image
                        ;       starting at
                        ;       3880h + bias back to tracks 0 and 1
                        ;       start this program at the next page boundary
0200                            org  ($+0100h) and 0ff00h

                        put$sys:
0200 318033                     lxi  sp,ccp-0080h   ; convenient place
0203 218033                     lxi  h,ccp-0080h    ; start of dump
0206 0600                       mvi  b,0            ; start with track
                        wr$trk:
0208 0e01                       mvi  c,1            ; start with sector
                        wr$sec:
020a cd0004                     call write$sec      ; write one sector
020d 118000                     lxi  d,128          ; length of each
0210 19                         dad  d              ; <hl>=<hl> + 128
0211 0c                         inr  c              ; <c> =<c> + 1
0212 79                         mov a,c             ; see if
0213 felb                       cpi  27             ; past end of track
0215 da0a02                     jc   wr$sec         ; no, do another

                        ; arrive here at end of track, move to next track

0218 04                         inr  b              ; track = track+1
0219 78                         mov a,b             ; see if
021a fe02                       cpi  2              ;  last track
021c da0802                     jc   wr$trk         ; no, do another

                        ;       done with putsys, halt for lack of anything
                        ;       better

021f fb                         ei
0220 76                         hlt
```


ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
                        ; user supplied subroutines for sector read and write

                        ;       move to next page boundary

0300                            org  ($+0100h) and 0ff00h

                        read$sec:
                                ; read the next sector
                                ; track in <b>,
                                ; sector in <c>
                                ; dmaaddr in <hl>

0300 c5                         push b
0301 e5                         push h

                        ; user defined read operation goes here
0302                            ds   64

0342 el                         pop  h
0343 cl                         pop  b
0344 c9                         ret

0400                            org  ($+0100h) and 0ff00h ;another page
                                                          ;boundary

                        write$sec:

                                ; same parameters as read$sec

0400 c5                         push b
0401 e5                         push h

                        ; user defined write operation goes here
0402                            ds   64

0442 el                         pop  h
0443 cl                         pop  b
0444 c9                         ret

                        ; end of getsys/putsys program

0445                            end
```

Appendix D: The MDS-800 Cold Start Loader for CP/M 2

```
                        title   mds cold start loader at 3000h'
                ;
                ;       mds-800 cold start loader for cp/m 2.0
                ;
                ;       version 2.0 august, 1979
                ;
0000 =          false   equ     0
ffff            true    equ     not false
0000 =          testing equ     false       if true, then go to mon80 on errors
                ;
                        if      testing
                bias    equ     03400h
                        endif
                        if      not testing
0000 =          bias    equ     0000h
                        endif
0000 =          cpmb    equ     bias            ;base of dos load
0806 =          bdos    equ     806h+bias       ;entry to dos for calls
1880 =          bdose   equ     1880h+bias      ;end of dos load
1600 =          boot    equ     1600h+bias      ;cold start entry point
1603 =          rboot   equ     boot+3          ;warm start entry point
                ;
3000                    org     03000h          ;loaded down from hardware boot at 3000H
                ;
1880 =          bdosl   equ     bdose-cpmb
0002 =          ntrks   equ     2               ;number of tracks to read
0031 =          bdoss   equ     bdosl/128       ;number of sectors in dos
0019 =          bdoso   equ     25              ;number of bdos sectors on track 0
0018 =          bdos1   equ     bdoss-bdoso     ;number of sectors on track 1
```

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
30                          ;
31      f800 =              mon80   equ     of800h  ;intel monitor base
32      ffof =              rmon80  equ     offofh  ;restart location for mon80
33      0078 =              base    equ     078h    ;'base' used by controller
34      0079 =              rtype   equ     base+1  ;result type
35      007b =              rbyte   equ     base+3  ;result byte
36      007f =              reset   equ     base+7  ;reset controller
37                          ;
38      0078 =              dstat   equ     base    ;disk status port
39      0079 =              ilow    equ     base+1  ;low iopb address
40      007a =              ihigh   equ     base+2  ;high iopb address
41      00ff =              bsw     equ     offh    ;boot switch
42      0003 =              recal   equ     3h      ;recalibrate selected drive
43      0004 =              readf   equ     4h      ;disk read function
44      0100 =              stack   equ     100h    ;use end of boot for stack
45                          ;
46                          rstart:
47      3000 310001                 lxi     sp,stack; ;in case of call to mon80
48                          ;       clear disk status
49      3003 db79                   in      rtype
50      3005 db7b                   in      rbyte
51                          ;       check if boot switch is off
52                          coldstart:
53      3007 dbff                   in      bsw
54      3009 e602                   ani     02h     ;switch on?
55      300b c20730                 jnz     coldstart
56                          ;       clear the controller
57      300e d37f                   out     reset   ;logic cleared
58                          ;
59                          ;
60      3010 0602                   mvi     b,ntrks ;number of tracks to read
61      3012 214230                 lxi     h,iopbo
62                          ;
```

```
63                      start:
64                      ;
65                      ;       read first/next track into cpmb
66      3015 7d                 mov     a,l
67      3016 d379               out     ilow
68      3018 7c                 mov     a,h
69      3019 d37a               out     ihigh
70      301b db78       waito:  in      dstat
71      301d e604               ani     4
72      301f ca1b30             jz      waito
73                      ;
74                      ;       check disk status
75      3022 db79               in      rtype
76      3024 e603               ani     11b
77      3026 fe02               cpi     2
78                      ;
79                              if      testing
80                              cnc     rmon80  ;go to monitor if 11 or 10
81                              endif
82                              if      not testing
83      3028 d20030             jnc     rstart  ;retry the load
84                              endif
85                      ;
86      302b db7b               in      rbyte   ;i/o complete, check status
87                      ;       if not ready, then go to mon80
88      302d 17                 ral
89      302e dc0fff             cc      rmon80  ;not ready bit set
90      3031 1f                 rar             ;restore
91      3032 e61e               ani     11110b  ;overrun/addr err/seek/crc/xxxx
92                      ;
93                              if      testing
94                              cnz     rmon80  ;go to monitor
95                              endif
96                              if      not testing
```

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
 97    3034 c20030             jnz     rstart      ;retry the load
 98                            endif
 99                    ;
100                    ;
101    3037 110700             lxi     d,iopbl     ;length of iopb
102    303a 19                 dad     d           ;addressing next iopb
103    303b 05                 dcr     b           ;count down tracks
104    303c c21530             jnz     start
105                    ;
106                    ;
107                    ;       jmp to boot to print initial message, and set up jmps
108    303f c30016             jmp     boot
109                    ;
110                    ;       parameter blocks
111    3042 80         iopbo:  db      80h         ;iocw, no update
112    3043 04                 db      readf       ;read function
113    3044 19                 db      bdoso       ;# sectors to read on track 0
114    3045 00                 db      0           ;track 0
115    3046 02                 db      2           ;start with sector 2 on track 0
116    3047 0000               dw      cpmb        ;start at base of bdos
117    0007 =          iopbl   equ     $-iopbo
118                    ;
119    3049 80         iopb1:  db      80h
120    304a 04                 db      readf
121    304b 18                 db      bdos1       ;sectors to read on track 1
122    304c 01                 db      1           ;track 1
123    304d 01                 db      1           ;sector 1
124    304e 800c               dw      cmpb+bdos0*128 ;base of second read
125                    ;
126    3050                    end
```

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

base	0078	33#	34	35	36	38	39	40
bdos	0806	18#						
bdoso	0019	28#	29	113	124			
bdos1	0018	29#	121					
bdose	1880	19#	25					
bdosl	1880	25#	27					
bdoss	0031	27#	29					
bias	0000	12#	15#	17	18	19	20	
boot	1600	20#	21	108				
bsw	00ff	41#	53					
coldstart	3007	52#	55					
cpmb	0000	17#	25	116	124			
dstat	0078	38#	70					
false	0000	7#	8	9				
ihigh	007a	40#	69					
ilow	0079	39#	67					
iopbo	3042	61	111#	117				
iopb1	3049	119#						
iopbl	0007	101	117#					
mon80	f800	31#						
ntrks	0002	26#	60					
rboot	1603	21#						
rbyte	007b	35#	50	86				
readf	0004	43#	112	120				
recal	0003	42#						
reset	007f	36#	57					
rmon80	ff0f	32#	80	89	94			
rstart	3000	46#	83	97				
rtype	0079	34#	49	75				
stack	0100	44#	47					
start	3015	63#	104					
testing	0000	9#	11	14	79	82	93	96
true	ffff	8#						
waito	301b	70#	72					

Appendix E: A Skeletal Cold Start Loader

```
                ; this is a sample cold start loader, which, when
                ; modified
                ; resides on track 00, sector 01 (the first sector on the
                ; diskette). we assume that the controller has loaded
                ; this sector into memory upon system start-up (this
                ; program can be keyed-in, or can exist in read/only
                ; memory
                ; beyond the address space of the cp/m version you are
                ; running). the cold start loader brings the cp/m system
                ; into memory at "loadp" (3400h + "bias"). in a 20k
                ; memory system, the value of "bias" is 0000h, with
                ; large
                ; values for increased memory sizes (see section 2).
                ; after
                ; loading the cp/m system, the cold start loader
                ; branches
                ; to the "boot" entry point of the bios, which begins at
                ; "bios" + "bias." the cold start loader is not used un-
                ; til the system is powered up again, as long as the bios
                ; is not overwritten. the origin is assumed at 0000h, an
                ; must be changed if the controller brings the cold start
                ; loader into another area, or if a read/only memory
                ; area
                ; is used.

0000                    org  0          ; base of ram in
                                        ; cp/m

0014 =          msize   equ  20         ; min mem size in
                                        ; kbytes
```

```
0000 =                  bias    equ  (msize-20)*1024    ; offset from 20k
                                                        ; system
3400 =                  ccp     equ  3400h+bias         ; base of the ccp
4a00 =                  bios    equ  ccp+1600h          ; base of the bios
0300 =                  biosl   equ  0300h              ; length of the bios
4a00 =                  boot    equ  bios
1900 =                  size    equ  bios+biosl-ccp     ; size of cp/m
                                                        ; system
0032 =                  sects   equ  size/128           ; # of sectors to load

                        ;       begin the load operation

                        cold:
0000 010200                     lxi  b,2                ; b=0, c=sector 2
0003 1632                       mvi  d,sects            ; d=# sectors to
                                                        ; load
0005 210034                     lxi  h,ccp              ; base transfer
                                                        ; address
                        lsect:  ; load the next sector

                        ;       insert inline code at this point to
                        ;       read one 128 byte sector from the
                        ;       track given in register b, sector
                        ;       given in register c,
                        ;       into the address given by <hl>
                        ; branch to location "cold" if a read error occurs
                        ;
                        ;
                        ;           user supplied read operation goes
                        ;           here...
                        ;
                        ;

0008 c36b00                     jmp  past$patch         ; remove this
                                                        ; when patched
000b                            ds   60h

                        past$patch:
                        ; go to next sector if load is incomplete
006b 15                         dcr  d                  ; sects=sects-1
006c ca004a                     jz   boot               ; head for the bios

                        ;       more sectors to load
                        ;
                        ; we aren't using a stack, so use <sp> as scratch
                        ; register
                        ;       to hold the load address increment

006f 318000                     lxi  sp,128             ; 128 bytes per
                                                        ; sector
0072 39                         dad  sp                 ; <hl> = <hl> +
                                                        128
```


ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
0073 0c                 inr   c           ; sector = sector + 1
0074 79                 mov a,c
0075 felb               cpi   27          ; last sector of
                                          ; track?
0077 da0800             jc    lsect       ; no, go read
                                          ; another

                    ; end of track, increment to next track

007a 0e01               mvi   c,l         ; sector = 1
007c 04                 inr   b           ; track = track + 1
007d c30800             jmp   lsect       ; for another group
0080                    end               ; of boot loader
```

Appendix F: CP/M Disk Definition Library

```
1: ;       CP/M 2.0 disk re-definition library
2: ;
3: ;       Copyright © 1979
4: ;       Digital Research
5: ;       Box 579
6: ;       Pacific Grove, CA
7: ;       93950
8: ;
9: ;       CP/M logical disk drives are defined using the
10: ;      macros given below, where the sequence of calls
11: ;      is:
12: ;
13: ;      disks   n
14: ;      diskdef parameter-list-0
15: ;      diskdef parameter-list-1
16: ;      ...
17: ;      diskdef parameter-list-n
18: ;      endef
19: ;
20: ;      where n is the number of logical disk drives attached
21: ;      to the CP/M system, and parameter-list-i defines the
22: ;      characteristics of the ith drive (i=0,1,...,n-1)
23: ;
24: ;      each parameter-list-i takes the form
25: ;              dn,fsc,lsc,[skf],bls,dks,dir,cks,ofs,[0]
26: ;      where
27: ;      dn      is the disk number 0,1,...,n-1
28: ;      fsc     is the first sector number (usually 0 or 1)
29: ;      lsc     is the last sector number on a track
30: ;      skf     is optional "skew factor" for sector translate
31: ;      bls     is the data block size (1024,2048,...,16384)
```

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
32: ;                   dks     is the disk size in bls increments (word)
33: ;                   dir     is the number of directory elements (word)
34: ;                   cks     is the number of dir elements to checksum
35: ;                   ofs     is the number of tracks to skip (word)
36: ;                   [0]     is an optional 0 which forces 16K/directory end
37: ;
38: ;                   for convenience, the form
39: ;                           dn,dm
40: ;                   defines disk dn as having the same characteristics as
41: ;                   a previously defined disk dm.
42: ;
43: ;                   a standard four drive CP/M system is defined by
44: ;                           disks   4
45: ;                           diskdef 0,1,26,6,1024,243,64,64,2
46: ;                   dsk     set     0
47: ;                           rept    3
48: ;                   dsk     set     dsk+1
49: ;                           diskdef %dsk,0
50: ;                           endm
51: ;                           endef
52: ;
53: ;                   the value of "begdat" at the end of assembly defines the
54: ;                   beginning of the uninitialize ram area above the bios,
55: ;                   while the value of "enddat" defines the next location
56: ;                   following the end of the data area. the size of this
57: ;                   area is given by the value of "datsiz" at the end of the
58: ;                   assembly. note that the allocation vector will be quite
59: ;                   large if a large disk size is defined with a small block
60: ;                   size.
61: ;
62: dskhdr             macro   dn
63: ;;                  define a single disk header list
64: dpe&dn:            dw      xlt&dn,0000h    ;translate table
```

```
65:                     dw      0000h,0000h     ;scratch area
66:                     dw      dirbuf,dpb&dn   ;dir buff,parm block
67:                     dw      csv&dn,alv&dn   ;check, alloc vectors
68:                     endm
69: ;
70: disks               macro   nd
71: ;;                  define nd disks
72: ndisks              set     nd              ;; for later reference
73: dpbase              equ     $               ;base of disk parameter blocks
74: ;;                  generate the nd elements
75: dsknxt              set     0
76:                     rept    nd
77:                     dskhdr  %dsknxt
78: dsknxt              set     dsknxc+1
79:                     endm
80:                     endm
81: ;
82: dpbhdr              macro   dn
83: dpb&dn              equ     $               ;disk parm block
84:                     endm
85: ;
86: ddb                 macro   data,comment
87: ;;                  define a db statement
88:                     db      data            comment
89:                     endm
90: ;
91: ddw                 macro   data,comment
92: ;;                  define a dw statement
93:                     dw      data            comment
94:                     endm
95: ;
96: gcd                 macro   m,n
97: ;;                  greatest common divisor of m,n
98: ;;                  produces value gcdn as result
```

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
 99: ;;          (used in sector translate table generation)
100: gcdm        set     m                   ;;variable for m
101: gcdn        set     n                   ;;variable for n
102: gcdr        set     0                   ;;variable for r
103:             rept    65535
104: gcdx        set     gcdm/gcdn
105: gcdr        set     gcdm-gcdx*gcdn
106:             if      gcdr = 0
107:             exitm
108:             endif
109: gcdm        set     gcdn
110: gcdn        set     gcdr
111:             endm
112:             endm
113: ;
114: diskdef     macro   dn,fsc,lsc,skf,bls,dks,dir,cks,ofs,k16
115: ;;          generate the set statements for later tables
116:             if      nul lsc
117: ;;          current disk dn             same as previous fsc
118: dpb&dn      equ     dpb&fsc             ;equivalent parameters
119: als&dn      equ     als&fsc             ;same allocation vector size
120: css&dn      equ     css&fsc             ;same checksum vector size
121: xlt&dn      equ     xlt&fsc             ;same translate table
122:             else
123: secmax       set     lsc-(fsc)           ;;sectors 0...secmax
124: sectors     set     secmax+1            ;;number of sectors
125: als&dn      set     (dks)/8             ;;size of allocation vector
126:             if      ((dks) mod 8) ne 0
127: als&dn      set     als&dn+1
128:             endif
129: css&dn      set     (cks)/4             ;;number of checksum elements
130: ;;          generate the block shift value
131: blkval      set     bls/128             ;;number of sectors/ block
```

```
132: blkshf      set     0               ;;counts right 0's in blkval
133: blkmsk      set     0               ;;fills with 1's from right
134:             rept    16              ;;once for each bit position
135:             if      blkval=1
136:             exitm
137:             endif
138: ;;          otherwise, high order 1 not found yet
139: blkshf      set     blkshf+1
140: blkmsk      set     (blkmsk shl 1) or 1
141: blkval      set     blkval/2
142:             endm
143: ;;          generate the extent mask byte
144: blkval      set     bls/1024        ;;number of kilobytes/ block
145: extmsk      set     0               ;;fill from right with 1's
146:             rept    16
147:             if      blkval=1
148:             exitm
149:             endif
150: ;;          otherwise more to shift
151: extmsk      set     (extmsk shl 1) or 1
152: blkval      set     blkval/2
153:             endm
154: ;;          may be double byte allocation
155:             if      (dks) > 256
156: extmsk      set     (extmsk shr 1)
157:             endif
158: ;;          may be optional [0] in last position
159:             if      not nul k16
160: extmsk      set     k16
161:             endif
162: ;;          now generate directory reservation bit vector
163: dirrem      set     dir             ;;# remaining to process
164: dirbks      set     bls/32          ;;number of entries per block
165: dirblk      set     0               ;;fill with 1's on each loop
166:             rept    16
167:             if      dirrem=0
168:             exitm
169:             endif
```

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
170: ;;                 not complete, iterate once again
171: ;;                 shift right and add 1 high order bit
172: dirblk             set     (dirblk shr l) or 8000h
173:                    if      dirrem > dirbks
174: dirrem             set     dirrem-dirbks
175:                    else
176: direem             set     0
177:                    endif
178:                    endm
179:                    dpbhdr  dn                      ;;generate equ $
180:                    ddw     %sectors,<;sec per track>
181:                    ddb     %blkshf,<;block shift>
182:                    ddb     %blkmsk,<;block mask>
183:                    ddb     %extmsk,<;extnt mask>
184:                    ddw     %(dks)-1,<;disk size-1>
185:                    ddw     %(dir)-1,<;directory max>
186:                    ddb     %dirblk shr 8,<;alloc0>
187:                    ddb     %dirblk and 0ffh,<;allocl>
188:                    ddw     %(cks)/4,<;check size>
189:                    ddw     %ofs,<;offset>
190: ;;                 generate the translate table, if requested
191:                    if      nul skf
192: xlt&dn             equ     0                       ;no xlate table
193:                    else
194:                    if      skf = 0
195: xlt&dn             equ     0                       ;no xlate table
196:                    else
197: ;;                 generate the translate table
198: nxtsec             set     0                       ;;next sector to fill
199: nxtbas             set     0                       ;;moves by one on overflow
200:                    gcd     %sectors,skf
201: ;;                 gcdn = gcd(sectors,skew)
202: neltst             set     sectors/gcdn
203: ;;                 neltst is number of elements to generate
```

```
204: ;;                 before we overlap previous elements
205: neltst             set     neltst          ;;counter
206: xlt&dn             equ     $               ;translate table
207:                    rept    sectors         ;;once for each sector
208:                    if      sectors < 256
209:                    ddb     %nxtsec+(fsc)
210:                    else
211:                    ddw     %nxtsec+(fsc)
212:                    endif
213: nxtsec             set     nxtsec+(skf)
214:                    if      nxtsec >= sectors
215: nxtsec             set     nxtsec-sectors
216:                    endif
217: neltst             set     neltst-1
218:                    if      neltst = 0
219: nxtbas             set     nxtbas+1
220: nxtsec             set     nxtbas
221: neltst             set     neltst
222:                    endif
223:                    endm
224:                    endif   ;;end of nul fac test
225:                    endif   ;;end of nul bls test
226:                    endm
227: ;
228: defds              macro   lab,space
229: lab:                ds      space
230:                    endm
231: ;
232: lds                macro   lb,dn,val
233:                    defds   lb&dn,%val&dn
234:                    endm
235: ;
236: endef              macro
237: ;;                generate the necessary ram data areas
```

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
begdat  equ     $
dirbuf: ds      128             ;directory access buffer
dsknxt  set     0
        rept    ndisks          ;;once for each disk
        lds     alv,%dsknxt,als
        lds     csv,%dsknxt,ccs
dsknxt  set     dsknxt+1
        endm
enddat  equ     $
datsiz  equ     $-begdat
;;      db 0 at this point forces hex record
        endm
```

Appendix G: Blocking and Deblocking Algorithms

```
                ;
                ;
                ;       sector deblocking algorithms for cp/m 2.0
                ;
                ;
                ;
                ;       utility macro to compute sector mask
                smask   macro   hblk
                ;;      compute log2(hblk), return @x as result
                ;;      (2 ** @x = hblk on return)
                @y      set     hblk
                @x      set     0
                ;;      count right shifts of @y until = 1
                        rept    8
                        if      @y = 1
                        exitm
                        endif
                ;;      @y is not 1, shift right one position
                @y      set     @y shr 1
                @x      set     @x + 1
                        endm
                        endm
                ;
                ;
                ;
                ;       cp/m to host disk constants
                ;
                ;
0800 =          blksiz  equ     2048            ;cp/m allocation size
0200 =          hstsiz  equ     512             ;host disk sector size
0014 =          hstspt  equ     20              ;host disk sectors/trk
0004 =          hstblk  equ     hstsiz/128      ;cp/m sects/host buff
0050 =          cpmspt  equ     hstblk * hstspt ;cp/m sectors/track
```

```
34      0003 =  secmsk  equ     hstblk-1        ;sector mask
35                      smask   hstblk          ;compute sector mask
36      0002 =  secshf  equ     @x              ;log2(hstblk)
37              ;
38              ;
39              ;
40              ;       bdos constants on entry to write
41              ;
42              ;
43      0000 =  wrall   equ     0               ;write to allocated
44      0001 =  wrdir   equ     1               ;write to directory
45      0002 =  wrual   equ     2               ;write to unallocated
46              ;
47              ;
48              ;
49              ;       the bdos entry points given below show the
50              ;       code which is relevant to deblocking only.
51              ;
52              ;
53              ;
54              ;       diskdef macro, or hand coded tables go here
55      0000 =  dpbase  equ     $               ;disk param block base
56              ;
57              boot:
58              wboot:
59                      ;enter here on system boot to initialize
60      0000 af         xra     a               ;0 to accumulator
61      0001 326a01     sta     hstact          ;host buffer inactive
62      0004 326c01     sta     unacnt          ;clear unalloc count
63      0007 c9         ret
64              ;
65              home:
66                      ;home the selected disk
67              home:
68      0008 3a6b01     lda     hstwrt          ;check for pending write
69      000b b7         ora     a
70      000c c21200     jnz     homed
71      000f 326a01     sta     hstact          ;clear host active flag
72              homed:
73      0012 c9         ret
74              ;
75              seldsk:
76                      ;select disk
77      0013 79         mov     a,c             ;selected disk number
78      0014 326101     sta     sekdsk          ;seek disk number
79      0017 6f         mov     l,a             ;disk number to hl
80      0018 2600       mvi     h,0
81                      rept    4               ;multiply by 16
82                      dad     h
83                      endm
84      001a+29         dad     h
85      001b+29         dad     h
86      001c+29         dad     h
87      001d+29         dad     h
88      001e 110000     lxi     d,dpbase        ;base of parm block
```


ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

```
 89    0021 19           dad     d               ;hl=.dpb(curdsk)
 90    0022 c9           ret
 91              ;
 92              settrk:
 93                        ;set track given by registers bc
 94    0023 60           mov     h,b
 95    0024 69           mov     l,c
 96    0025 226201       shld    sektrk          ;track to seek
 97    0028 c9           ret
 98              ;
 99              setsec:
100                        ;set sector given by register c
101    0029 79           mov     a,c
102    002a 326401       sta     seksec          ;sector to seek
103    002d c9           ret
104             ;
105             setdma:
106                        ;set dma address given by bc
107    002e 60           mov     h,b
108    002f 69           mov     l,c
109    0030 227501       shld    dmaadr
110    0033 c9           ret
111             ;
112             sectran:
113                        ;translate sector number bc
114    0034 60           mov     h,b
115    0035 69           mov     l,c
116    0036 c9           ret
117             ;
118             ;
119             ;
120             ;         the read entry point takes the place of
121             ;         the previous bios definition for read.
122             ;
123             ;
124             read:
125                        ;read the selected cp/m sector
126    0037 af           xra     a
127    0038 326c01       sta     unacnt
128    003b 3e01         mvi     a,1
129    003d 327301       sta     readop          ;read operation
130    0040 327201       sta     rsflag          ;must read data
131    0043 3e02         mvi     a,wrual
132    0045 327401       sta     wrtype          ;treat as unalloc
133    0048 c3b600       jmp     rwoper          ;to perform the read
134             ;
135             ;
136             ;
137             ;         the write entry point takes the place of
138             ;         the previous bios definition for write.
139             ;
140             ;
141             write:
142                        ;write the selected cp/m sector
143    004b af           xra     a               ;0 to accumulator
```

```
144     004c 327301         sta     readop          ;not a read operation
145     004f 79             mov     a,c             ;write type in c
146     0050 327401         sta     wrtype
147     0053 fe02           cpi     wrual           ;write unallocated?
148     0055 c26f00         jnz     chkuna          ;check for unalloc
149                 ;
150                 ;       write to unallocated, set parameters
151     0058 3e10           mvi     a,blksiz/128    ;next unalloc recs
152     005a 326c01         sta     unacnt
153     005d 3a6101         lda     sekdsk          ;disk to seek
154     0060 326d01         sta     unadsk          ;unadsk = sekdsk
155     0063 2a6201         lhld    sektrk
156     0066 226e01         shld    unatrk          ;unatrk = sectrk
157     0069 3a6401         lda     seksec
158     006c 327001         sta     unasec          ;unasec = seksec
159                 ;
160                 chkuna:
161                         ;check for write to unallocated sector
162     006f 3a6c01         lda     unacnt          ;any unalloc remain?
163     0072 b7             ora     a
164     0073 caae00         jz      alloc           ;skip if not
165                 ;
166                 ;       more unallocated records remain
167     0076 3d             dcr     a               ;unacnt = unacnt-1
168     0077 326c01         sta     unacnt
169     007a 3a6101         lda     sekdsk          ;same disk?
170     007d 216d01         lxi     h,unadsk
171     0080 be             cmp     m               ;sekdsk = unadsk?
172     0081 c2ae00         jnz     alloc           ;skip if not
173                 ;
174                 ;       disks are the same
175     0084 216e01         lxi     h,unatrk
176     0087 cd5301         call    sektrkcmp       ;sektrk = unatrk?
177     008a c2ae00         jnz     alloc           ;skip if not
178                 ;
179                 ;       tracks are the same
180     008d 3a6401         lda     seksec          ;same sector?
181     0090 217001         lxi     h,unasec
182     0093 be             cmp     m               ;seksec = unasec?
183     0094 c2ae00         jnz     alloc           ;skip if not
184                 ;
185                 ;       match, move to next sector for future ref
186     0097 34             inr     m               ;unasec = unasec+1
187     0098 7e             mov     a,m             ;end of track?
188     0099 fe50           cpi     cpmspt          ;count cp/m sectors
189     009b daa700         jc      noovf           ;skip if no overflow
190                 ;
191                 ;       overflow to next track
192     009e 3600           mvi     m,o             ;unasec = o
193     00a0 2a6e01         lhld    unatrk
194     00a3 23             inx     h
195     00a4 226e01         shld    unatrk          ;unatrk = unatrk+1
196                 ;
197                 noovf:
198                         ;match found, mark as unnecessary read
199     00a7 af             xra     a               ;0 to accumulator
```

```
200     00ab 327201         sta     rsflag          ;rsflag = 0
201     00ab c3b600         jmp     rwoper          ;to perform the write
202                 ;
203                 alloc:
204                         ;not an unallocated record, requires pre-read
205     00ae af             xra     a               ;0 to accum
206     00af 326c01         sta     unacnt          ;unacnt = 0
207     00b2 3c             inr     a               ;1 to accum
208     00b3 327201         sta     rsflag  = 1     ;rsflag = 1
209                 ;
210                 ;
211                 ;
212                 ;       common code for read and write follows
213                 ;
214                 ;
215                 rwoper:
216                         ;enter here to perform the read/write
217     00b6 af             xra     a               ;zero to accum
218     00b7 327101         sta     erflag          ;no errors (yet)
219     00ba 3a6401         lda     seksec          ;compute host sector
220                         rept    secshf
221                         ora     a               ;carry = 0
222                         rar                     ;shift right
223                         endm
224     00bd+b7             ora     a               ;carry = 0
225     00be+1f             rar                     ;shift right
226     00bf+b7             ora     a               ;carry = 0
227     00c0+1f             rar                     ;shift right
228     00c1 326901         sta     sekhst          ;host sector to seek
229                 ;
230                 ;       active host sector?
231     00c4 216a01         lxi     h,hstact        ;host active flag
232     00c7 7e             mov     a,m
233     00c8 3601           mvi     m,1             ;always becomes 1
234     00ca b7             ora     a               ;was it already?
235     00cb caf200         jz      filhst          ;fill host if not
236                 ;
237                 ;       host buffer active, same as seek buffer?
238     00ce 3a6101         lda     sekdsk
239     00d1 216501         lxi     h,hstdsk        ;same disk?
240     00d4 be             cmp     m               ;sekdsk = hstdsk?
241     00d5 c2eb00         jnz     nomatch
242                 ;
243                 ;       same disk, same track?
244     00d8 216601         lxi     h,hsttrk
245     00db cd5301         call    sektrkcmp       ;sektrk = hsttrk?
246     00de c2eb00         jnz     nomatch
247                 ;
248                 ;       same disk, same track, same buffer?
249     00e1 3a6901         lda     sekhst
250     00e4 216801         lxi     h,hstsec        ;sekhst = hstsec?
251     00e7 be             cmp     m
252     00e8 ca0f01         jz      match           ;skip if match
253                 ;
254                 nomatch:
```

```
255                     ;proper disk, but not correct sector
256   00eb 3a6b01       lda     hstwrt          ;host written?
257   00ee b7           ora     a
258   00ef c45f01       cnz     writehst        ;clear host buff
259                     ;
260                   filhst:
261                     ;may have to fill the host buffer
262   00f2 3a6101       lda     sekdsk
263   00f5 326501       sta     hstdsk
264   00f8 2a6201       lhld    sektrk
265   00fb 226601       shld    hsttrk
266   00fe 3a6901       lda     sekhst
267   0101 326801       sta     hstsec
268   0104 3a7201       lda     rsflag          ;need to read?
269   0107 b7           ora     a
270   0108 c46001       cnz     readhst         ;yes, if 1
271   010b af           xra     a               ;0 to accum
272   010c 326b01       sta     hstwrt          ;no pending write
273                     ;
274                 match:
275                     ;copy data to or from buffer
276   010f 3a6401       lda     seksec          ;mask buffer number
277   0112 e603         ani     secmsk          ;least signif bits
278   0114 6f           mov     l,a             ;ready to shift
279   0115 2600         mvi     h,0             ;double count
280                     rept    7               ;shift left 7
281                     dad     h
282                     endm
283   0117+29           dad     h
284   0118+29           dad     h
285   0119+29           dad     h
286   011a+29           dad     h
287   011b+29           dad     h
288   011c+29           dad     h
289   011d+29           dad     h
290                 ;   hl has relative host buffer address
291   011e 117701       lxi     d,hstbuf
292   0121 19           dad     d               ;hl = host address
293   0122 eb           xchg                    ;now in de
294   0123 2a7501       lhld    dmaadr          ;get/put cp/m data
295   0126 0e80         mvi     c,128           ;length of move
296   0128 3a7301       lda     readop          ;which way?
297   012b b7           ora     a
298   012c c23501       jnz     rwmove          ;skip if read
299                 ;
300                 ;   write operation, mark and switch direction
301   012f 3e01         mvi     a,1
302   0131 326b01       sta     hstwrt          ;hstwrt = 1
303   0134 eb           xchg                    ;source/dest swap
304                 ;
305                 rwmove:
306                     ;c initially 128, de is source, hl is dest
307   0135 1a           ldax    d               ;source character
308   0136 13           inx     d
309   0137 77           mov     m,a             ;to dest
```

```
310     0138 23         inx     h
311     0139 od         dcr     c               ;loop 128 times
312     013a c23501     jnz     rwmove
313             ;
314             ;       data has been moved to/from host buffer
315     013d 3a7401     lda     wrtype          ;write type
316     0140 fe01       cpi     wrdir           ;to directory?
317     0142 3a7101     lda     erflag          ;in case of errors
318     0145 c0         rnz                     ;no further processing
319             ;
320             ;       clear host buffer for directory write
321     0146 b7         ora     a               ;errors?
322     0147 c0         rnz                     ;skip if so
323     0148 af         xra     a               ;0 to accum
324     0149 326b01     sta     hstwrt          ;buffer written
325     014c cd5f01     call    writehst
326     014f 3a7101     lda     erflag
327     0152 c9         ret
328             ;
329             ;
330             ;
331             ;       utility subroutine for 16-bit compare
332             ;
333             ;
334             sektrkcmp:
335                     ;hl = .unatrk or .hsttrk, compare with sektrk
336     0153 eb         xchg
337     0154 216201     lxi     h,sektrk
338     0157 1a         ldax    d               ;low byte compare
339     0158 be         cmp     m               ;same?
340     0159 c0         rnz                     ;return if not
341             ;       low bytes equal, test high 1s
342     015a 13         inx     d
343     015b 23         inx     h
344     015c 1a         ldax    d
345     015d be         cmp     m               ;sets flags
346     015e c9         ret
347             ;
348             ;
349             ;
350             ;       writehst performs the physical write to
351             ;       the host disk, readhst reads the physical
352             ;       disk.
353             ;
354             ;
355             writehst:
356                     ;hstdsk = host disk #, hsttrk = host track #,
357                     ;hstsec = host sect #. write "hstsiz" bytes
358                     ;from hstbuf and return error flag in erflag.
359                     ;return erflag non-zero if error
360     015f c9         ret
361             ;
362             readhst:
363                     ;hstdsk = host disk #, hsttrk = host track #,
364                     ;hstsec = host sect #. read "hstsiz" bytes
```

```
                        ;into hstbuf and return error flag in erflag.
0160 c9                 ret
                ;
                ;
                ;
                ;       uninitialized ram data areas
                ;
                ;
                ;
0161    sekdsk: ds      1               ;seek disk number
0162    sektrk: ds      2               ;seek track number
0164    seksec: ds      1               ;seek sector number
                ;
0165    hstdsk: ds      1               ;host disk number
0166    hsttrk: ds      2               ;host track number
0168    hstsec: ds      1               ;host sector number
                ;
0169    sekhst: ds      1               ;seek shr secshf
016a    hstact: ds      1               ;host active flag
016b    hstwrt: ds      1               ;host written flag
                ;
016c    unacnt: ds      1               ;unalloc rec cnt
016d    unadsk: ds      1               ;last unalloc disk
016e    unatrk: ds      2               ;last unalloc track
0170    unasec: ds      1               ;last unalloc sector
                ;
0171    erflag: ds      1               ;error reporting
0172    rsflag: ds      1               ;read sector flag
0173    readop: ds      1               ;1 if read operation
0174    wrtype: ds      1               ;write operation type
0175    dmaadr: ds      2               ;last dma address
0177    hstbuf: ds      hstsiz          ;host buffer
                ;
                ;
                ;
                ;       the endef macro invocation goes here
                ;
                ;
0377                    end
```


ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

alloc	00ae	164	172	177	183	203#		
blksiz	0800	29#	151					
boot	0000	57#						
chkuna	006f	148	160#					
cpmspt	0050	33#	188					
dmaadr	0175	109	294	395#				
dpbase	0000	55#	88					
erflag	0171	218	317	326	391#			
filhst	00f2	235	260#					
home	0008	65#	67#					
homed	0012	70	72#					
hstact	016a	61	71	231	383#			
hstblk	0004	32#	33	34	35			
hstbuf	0177	291	396#					
hstdsk	0165	239	263	378#				
hstsec	0168	250	267	380#				
hstsiz	0200	30#	32	396				
hstspt	0014	31#	33					
hsttrk	0166	244	265	379#				
hstwrt	016b	68	256	272	302	324	384#	
match	010f	252	274#					
nomatch	00eb	241	246	254#				
noovf	00a7	189	197#					
read	0037	124#						
readhst	0160	270	362#					
readop	0173	129	144	296	393#			
rsflag	0172	130	200	208	268	392#		
rwmove	0135	298	305#	312				
rwoper	00b6	133	201	215#				
secmsk	0003	34#	277					
secshf	0002	36#	220					
sectran	0034	112#						
sekdsk	0161	78	153	169	238	262	374#	
sekhst	0169	228	249	266	382#			
seksec	0164	102	157	180	219	276	376#	
sektrk	0162	96	155	264	337	375#		
sektrkcmp	0153	176	245	334#				
seldsk	0013	75#						
setdma	002e	105#						
setsec	0029	99#						
settrk	0023	92#						
unacnt	016c	62	127	152	162	168	206	386#
unadsk	016d	154	170	387#				
unasec	0170	158	181	389#				
unatrk	016e	156	175	193	195	388#		
wboot	0000	58#						
wrall	0000	43#						
wrdir	0001	44#	316					
write	004b	141#						
writehst	015f	258	325	355#				
wrtype	0174	132	146	315	394#			
wrual	0002	45#	131	147				

Appendix H: Glossary

address: Number representing the location of a byte in memory. Within CP/M there are two kinds of addresses: logical and physical. A physical address refers to an absolute and unique location within the computer's memory space. A logical address refers to the offset or displacement of a byte in relation to a base location. A standard CP/M program is loaded at address 0100H, the base value; the first instruction of a program has a physical address of 0100H and a relative address or offset of OH.

allocation vector (ALV): An allocation vector is maintained in the BIOS for each logged in disk drive. A vector consists of a string of bits, one for each block on the drive. The bit corresponding to a particular block is set to one when the block has been allocated and to zero otherwise. The first two bytes of this vector are initialized with the bytes AL0 and AL1 on, thus allocating the directory blocks. CP/M Function 27 returns the allocation vector address.

AL0, AL1: Two bytes in the disk parameter block that reserve data blocks for the directory. These two bytes are copied into the first two bytes of the allocation vector when a drive is logged in. (See allocation vector.)

ALV: See allocation vector.

ambiguous filename: Filename that contains either of the CP/M wildcard characters, ? or *, in the primary filename or the filetype, or both. When you replace characters in a filename with these wildcard characters, you create an ambiguous filename and can easily reference more than one CP/M file in a single command line.

American Standard Code for Information Interchange: See ASCII.

applications program: Program designed to solve a specific problem. Typical applications programs are business accounting packages, word processing (editing) programs and mailing list programs.

archive attribute: File attribute controlled by the high-order bit of the t3 byte (FCB+11) in a directory element. This attribute is set if the file has been archived.

argument: Symbol, usually a letter, indicating a place into which you can substitute a number, letter or name to give an appropriate meaning to the formula in question.

ASCII: American Standard Code for Information Interchange. ASCII is a standard set of seven-bit numeric character codes used to represent characters in memory. Each character requires one byte of memory with the high-order bit usually set to zero. Characters can be numbers, letters, and symbols. An ASCII file can be intelligibly displayed on the video screen or printed on paper.

assembler: Program that translates assembly language into the binary machine code. Assembly language is simply a set of mnemonics used to designate the instruction set of the CPU. (See ASM in Section 3 of this manual.)

back-up: Copy of a disk or file made for safekeeping, or the creation of the duplicate disk or file.

Basic Disk Operating System: See BDOS.

BDOS: Basic Disk Operating System. The BDOS module of the CP/M operating system provides an interface for a user program to the operating system. This interface is in the form of a set of function calls which may be made to the BDOS through calls to location 0005H in page zero. The user program specifies the number of the desired function in register C. User programs running under CP/M should use BDOS functions for all I/O operations to remain compatible with other CP/M systems and future releases. The BDOS normally resides in high memory directly below the BIOS.

bias: Address value which when added to the origin address of your BIOS module produces 1F80H, the address of the BIOS module in the MOVCPM image. There is also a bias value that when added to the BOOT module origin produces 0900H, the address of the BOOT module in the MOVCPM image. You must use these bias values with the R command under DDT or SID when you patch a CP/M system. If you do not, the patched system may fail to function.

binary: Base 2 numbering system. A binary digit can have one of two values: 0 or 1. Binary numbers are used in computers because the hardware can most easily exhibit two states: off and on. Generally, a bit in memory represents one binary digit.

Basic Input/Output System: See BIOS.

BIOS: Basic Input/Output System. The BIOS is the only hardware-dependent module of the CP/M system. It provides the BDOS with a set of primitive I/O operations. The BIOS is an assembly language module usually written by the user, hardware manufacturer or independent software vendor, and is the key to CP/M's portability. The BIOS interfaces the CP/M system to its hardware environment through a standardized jump table at the front of the BIOS routine and through a set of disk parameter tables which define the disk environment. Thus, the BIOS provides CP/M with a completely table-driven I/O system.

BIOS base: Lowest address of the BIOS module in memory, that by definition must be the first entry point in the BIOS jump table.

bit: Switch in memory that can be set to on (1) or off (0). Bits are grouped into bytes, eight bits to a byte, which is the smallest directly addressable unit in an Intel 8080 or Zilog Z-80. By common convention, the bits in a byte are numbered from right (0 for the low order bit) to left (7 for the high order bit). Bit values are often represented in hexadecimal notation by grouping the bits from the low order bit in groups of four. Each group of four bits can have a value from 0 to 15 and thus can easily be represented by one hexadecimal digit.

BLM: See block mask.

block: Basic unit of disk space allocation. Each disk drive has a fixed block size (BLS) defined in its disk parameter block in the BIOS. A block can consist of 1K, 2K, 4K, 8K or 16K consecutive bytes. Blocks are numbered relative to zero so that each block is unique and has a byte displacement in a file equal to the block number times the block size.

block mask (BLM): Byte value in the disk parameter block at DPB + 3. The block mask is always one less than the number of 128 byte sectors that are in one block. Note: BLM = (2 ** BSH) - 1.

block shift (BSH): Byte parameter in the disk parameter block at DPB + 2. Values for the block shift and block mask (BLM) are determined by the block size (BLS). Note: BLM = (2 ** BSH) - 1.

blocking & deblocking algorithm: In some disk subsystems the disk sector size is larger than 128 bytes, usually 256, 512, 1024 or 2048 bytes. When the host sector size is larger than 128 bytes, host sectors must be buffered in memory and the 128 byte CP/M sectors must be blocked and deblocked by adding an additional module, the blocking and deblocking algorithm, between the BIOS disk I/O routines and the actual disk I/O. The host sector size must be an even multiple of 128 bytes for the algorithm to work correctly. The blocking and deblocking algorithm allows the BDOS and BIOS to function exactly as if the entire disk consisted only of 128 byte sectors, as in the standard CP/M installation.

BLS: Block size in bytes. See block.

boot: Process of loading an operating system into memory. A boot program is a small piece of code that is automatically executed when you power-up or reset your computer. The boot program loads the rest of the operating system into memory in a manner similar to a person pulling himself up by his own bootstraps. This process is sometimes called a "cold boot" or "cold start." Bootstrap procedures vary from system to system. The boot program must be customized for the memory size and hardware environment that the operating system manages. Typically, the boot resides on the first sector of the system tracks on your system diskette. When executed, the boot loads the remaining sectors of the system tracks into high memory at the location for which the CP/M system has been configured. Finally, the boot transfers execution to the boot entry point in the BIOS jump table so that the system can initialize itself. In this case, the boot program should be placed at 900H in the SYSGEN image. Alternatively, the boot program may be located in ROM.

bootstrap: See boot.

BSH: See block shift.

BTREE: General purpose file access method that has become the standard organization for indexes in large data base systems. BTREE provides near optimum performance over the full range of file operations, such as insertion, deletion, search, and search next.

buffer: Area of memory that temporarily stores data during the transfer of information.

built-in commands: Commands that permanently reside in memory. They respond quickly because they are not accessed from a disk.

byte: Unit of memory or disk storage containing eight bits. A byte can represent a binary number between 0 and 255, and is the smallest unit of memory that can be addressed directly in 8 bit CPUs such as the Intel 8080 or Zilog Z-80.

CCP: Console Command Processor. The CCP is a module of the CP/M operating system. It is loaded directly below the BDOS module and interprets and executes commands typed by the console user. Usually these commands are programs that the CCP loads and calls. Upon completion, a command program may return control to the CCP if it has not overwritten it. If it has, the program can reload the CCP into memory by a warm boot operation initiated by either a jump to zero, BDOS system reset (function 0), or a cold boot. Except for its location in high memory, the CCP works like any other standard CP/M program; that is, it makes only BDOS function calls for its I/O operations.

CCP base: Lowest address of the CCP module in memory. This term sometimes refers to the base of the CP/M system in memory, as the CCP is normally the lowest CP/M module in high memory.

checksum vector (CSV): Contiguous data area in the BIOS, with one byte for each directory sector to be checked, i.e., CKS bytes. (See CKS.) A checksum vector is initialized and maintained for each logged in drive. Each directory access by the system results in a checksum calculation that is compared with the one in the checksum vector. If there is a discrepancy, the drive is set to read-only status. This feature prevents the user from inadvertently switching disks without logging in the new disk. If the new disk is not logged in, it is treated the same as the old one, and data on it may be destroyed if writing is done.

CKS: Number of directory records to be checked summed on directory accesses. This is a parameter in the disk parameter block located in the BIOS. If the value of CKS is zero, then no directory records are checked. CKS is also a parameter in the diskdef macro library, where it is the actual number of directory elements to be checked rather than the number of directory records.

cold boot: See boot. Cold boot also may refer to a jump to the boot entry point in the BIOS jump table.

COM: Filetype for a CP/M command file. See command file.

command: CP/M command line. In general, a CP/M command line has three parts: the command keyword, command tail, and a carriage return. To execute a command, enter a CP/M command line directly after the CP/M prompt at the console and press the carriage return or enter key.

command file: Executable program file of filetype COM. A command file is a machine language object module ready to be loaded and executed at the absolute address of 0100H. To execute a command file, enter its primary filename as the command keyword in a CP/M command line.

command keyword: Name that identifies a CP/M command, usually the primary filename of a file of type COM, or a built-in command. The command keyword precedes the command tail and the carriage return in the command line.

command syntax: Statement that defines the correct way to enter a command. The correct structure generally includes the command keyword, the command tail, and a carriage return. A syntax line usually contains symbols that you should replace with actual values when you enter the command.

command tail: Part of a command that follows the command keyword in the command line. The command tail can include a drive specification, a filename and/or filetype, and options or parameters. Some commands do not require a command tail.

CON: Mnemonic that represents the CP/M console device (see console). For example, the CP/M command "PIP CON:=TEST.SUB" displays the file TEST.SUB on the console device. The explanation of the STAT command tells how to assign the logical device CON: to various physical devices.

concatenate: Name of the PIP operation that copies two or more separate files into one new file in the specified sequence.

concurrency: Execution of two processes or operations simultaneously.

CONIN: BIOS entry point to a routine that reads a character from the console device.

CONOUT: BIOS entry point to a routine that sends a character to the console device.

console: Primary input/output device. The console consists of a listing device, such as a screen or teletype, and a keyboard through which the user communicates with the operating system or applications program.

Console Command Processor: See CCP.

CONST: BIOS entry point to a routine that returns the status of the console device.

control character: Nonprinting character combination. CP/M interprets some control characters as simple commands such as line editing functions. To enter a control character, hold down the CONTROL key and strike the specified character key.

Control Program for Microcomputers: See CP/M.

CP/M: Control Program for Microcomputers. An operating system that manages computer resources and provides a standard systems interface to software written for a large variety of microprocessor-based computer systems.

CP/M 1.4 compatibility: For a CP/M 2 system to be able to read correctly single density diskettes produced under a CP/M 1.4 system, the extent mask must be zero and the block size 1K. This is because under CP/M 2 an FCB may contain more than one extent. The number of extents that may be contained by an FCB is EXM+1. The issue of CP/M 1.4 compatibility also concerns random file I/O. To perform random file I/O under CP/M 1.4, you must maintain an FCB for each extent of the file. This scheme is upward compatible with CP/M 2 for files not exceeding 512K bytes, the largest file size supported under CP/M 1.4. If you wish to implement random I/O for files larger than 512K bytes under CP/M 2, you must use the random read and random write functions (BDOS functions 33, 34 and 36). In this case, only one FCB is used, and if CP/M 1.4 compatibility is required, the program must use the return version number function (BDOS function 12) to determine which method to employ.

CP/M prompt: Characters that indicate that CP/M is ready to execute your next command. The CP/M prompt consists of an upper-case letter (A-P) followed by a ">" character; for example, A>. The letter designates which drive is currently logged in as the default drive. CP/M will search this drive for the command file specified, unless the command is a built-in command or prefaced by a select drive command; for example, B:STAT.

CP/NET: Digital Research network operating system enabling microcomputers to obtain access to common resources via a network. CP/NET consists of MP/M masters and CP/M slaves with a network interface between them.

CSV: See checksum vector.

cursor: One-character symbol that can appear anywhere on the console screen. The cursor indicates the position where the next keystroke at the console will have an effect.

data file: File containing information that will be processed by a program.

deblocking: See blocking & deblocking algorithm.

default: Currently selected disk drive and user number. Any command that does not specify a disk drive or a user number references the default disk drive and user number. When CP/M is first invoked, the default disk drive is drive A, and the default user number is 0.

default buffer: Default 128-byte buffer maintained at 0080H in page zero. When the CCP loads a COM file, this buffer is initialized to the command tail; that is, any characters typed after the COM file name are loaded into the buffer. The first byte at 0080H contains the length of the command tail, while the command tail itself begins at 0081H. The command tail is terminated by a byte containing a binary zero value. The I command under DDT and SID initializes this buffer in the same way as the CCP.

default FCB: Two default FCBs are maintained by the CCP at 005CH and 006CH in page zero. The first default FCB is initialized from the first delimited field in the command tail, and the second default FCB is initialized from the next field in the command tail.

delimiter: Special characters that separate different items in a command line; for example, a colon separates the drive specification from the filename. The CCP recognizes the following characters as delimiters: . : = ; < > _ , blank, and carriage return. Several CP/M commands also treat the following as delimiter characters: , [ ] ( ) $. It is advisable to avoid the use of delimiter characters and lower-case characters in CP/M file names.

DIR: Parameter in the diskdef macro library that specifies the number of directory elements on the drive.

DIR attribute: File attribute. A file with the DIR attribute can be displayed by a DIR command. The file can be accessed from the default user number and drive only.

DIRBUF: 128-byte scratchpad area for directory operations, usually located at the end of the BIOS. DIRBUF is used by the BDOS during its directory operations. DIRBUF also refers to the two-byte address of this scratchpad buffer in the disk parameter header at DPbase + 8 bytes.

directory: Portion of a disk that contains entries for each file on the disk. In response to the DIR command, CP/M displays the filenames stored in the directory. The directory also contains the locations of the blocks allocated to the files. Each file directory element is in the form of a 32-byte FCB, although one file may have several elements, depending on its size. The maximum number of directory elements supported is specified by the drive's disk parameter block value for DRM.

directory element: Data structure. Each file on a disk has one or more 32-byte directory elements associated with it. There are four directory elements per directory sector. Directory elements may also be referred to as directory FCBs.

directory entry: File entry displayed by the DIR command. Sometimes this term may refer to a physical directory element.

disk, diskette: Magnetic media used for mass storage in a computer system. Programs and data are recorded on the disk in the same way music can be recorded on cassette tape. The CP/M operating system must be initially loaded from disk when the computer is turned on. Diskette refers to smaller capacity removable floppy diskettes, while disk may refer to either a diskette, removable cartridge disk or fixed hard disk. Hard disk capacities range from five to several hundred megabytes of storage.

diskdef macro library: Library of code that when used with MAC (the Digital Research macro assembler) creates disk definition tables such as the DPB and DPH automatically.

disk drive: Peripheral device that reads and writes information on disk or diskettes. CP/M assigns a letter to each drive under its control. For example, CP/M may refer to the drives in a four-drive system as A, B, C, and D.

disk parameter block (DPB): Data structure referenced by one or more disk parameter headers. The disk parameter block defines disk characteristics in the fields listed below:

SPT	The total number of sectors per track
BSH	The data allocation block shift factor
BLM	The data allocation block mask
EXM	The extent mask determined by BLS and DSM
DSM	The maximum data block number
DRM	Maximum number of directory entries—1
AL0	Reserves directory blocks
AL1	Reserves directory blocks
CKS	The number of directory sectors check summed
OFF	The number of reserved system tracks

The address of the disk parameter block is located in the disk parameter header at DPbase + 0AH. CP/M Function 31 returns the DPB address. Drives with the same characteristics may use the same disk parameter header, and thus the same DPB. However, drives with different characteristics must each have their own disk parameter header and disk parameter blocks. When the BDOS calls the SELDSK entry point in the BIOS, SELDSK must return the address of the drive's disk parameter header in registers HL.

disk parameter header (DPH): Data structure that contains information about the disk drive and provides a scratchpad area for certain BDOS operations. The disk parameter header contains six bytes of scratchpad area for the BDOS, and the following five two-byte parameters:

XLT	The sector translation table address
DIRBUF	Directory buffer address
DPB	Disk parameter block address
CSV	Checksum vector address
ALV	Allocation vector address

Given n disk drives, the disk parameter headers are arranged in a table whose first row of 16 bytes corresponds to drive 0, with the last row corresponding to drive n-1.

DKS: Parameter in the diskdef macro library specifying the number of data blocks on the drive.

DMA: Direct memory access. DMA is a method of transferring data from the disk into memory directly. In a CP/M system, the BDOS calls the BIOS entry point READ to read a sector from the disk into the currently selected DMA address. The DMA address must be

the address of a 128-byte buffer in memory, either the default buffer at 0080H in page zero, or a user-assigned buffer in the TPA. Similarly, the BDOS calls the BIOS entry point WRITE to write the record at the current DMA address to the disk.

DN: Parameter in the diskdef macro library specifying the logical drive number.

DPB: See disk parameter block.

DPH: See disk parameter header.

DRM: 2-byte parameter in the disk parameter block at DPB + 7. DRM is one less than the total number of directory entries allowed for the drive. This value is related to DPB bytes AL0 and AL1, which allocate up to 16 blocks for directory entries.

DSM: 2-byte parameter of the disk parameter block at DPB + 5. DSM is the maximum data block number supported by the drive. The product BLS times (DSM+1) is the total number of bytes held by the drive. This must not exceed the capacity of the physical disk less the reserved system tracks.

editor: Utility program that creates and modifies text files. An editor can be used for creation of documents or, creation of code for computer programs. The CP/M editor is invoked by typing the command ED next to the system prompt on the console.

EX: Extent number field in an FCB. See extent.

executable: Ready to be run by the computer. Executable code is a series of instructions that can be carried out by the computer. For example, the computer cannot execute names and addresses, but it can execute a program that prints all those names and addresses on mailing labels.

execute a program: Start the processing of executable code.

EXM: See extent mask.

extent: 16K consecutive bytes in a file. Extents are numbered from 0 to 31. One extent may contain 1, 2, 4, 8 or 16 blocks. EX is the extent number field of an FCB and is a one byte field at FCB + 12, where FCB labels the first byte in the FCB. Depending on the block size (BLS) and the maximum data block number (DSM), an FCB may contain 1, 2, 4, 8 or 16 extents. The EX field is normally set to 0 by the user but contains the current extent number during file I/O. The term FCB folding describes FCBs containing more than one extent. In CP/M version 1.4, each FCB contained only one extent. Users attempting to perform random record I/O and maintain CP/M 1.4 compatibility should be aware of the implications of this difference. See CP/M 1.4 compatibility.

extent mask (EXM): A byte parameter in the disk parameter block located at DPB + 3. The value of EXM is determined by the block size (BLS) and whether the maximum data block number (DSM) exceeds 255. There are EXM + 1 extents per directory FCB.

FCB: See file control block.

file: Collection of characters, instructions, or data that can be referenced by a unique identifier. Files are usually stored on various types of media, such as disks, diskettes, or magnetic tape. A CP/M file is identified by a file specification and resides on disk as a collection of from zero to 65,536 records. Each record is 128 bytes and can contain either binary or ASCII data. Binary files contain bytes of data that can vary in value from 0H to

0FFH. ASCII files contain sequences of character codes delineated by a carriage return-line feed combination; normally byte values range from 0H to 7FH. The directory maps the file as a series of physical blocks. Although files are defined as a sequence of consecutive logical records, these records may not reside in consecutive sectors on the disk. (see also block, directory, extent, record, sector).

file control block (FCB): Structure used for accessing files on disk. Contains the drive, filename, filetype, and other information describing a file to be accessed or created on the disk. A file control block consists of 36 consecutive bytes specified by the user for file I/O functions. FCB can also refer to a directory element in the directory portion of the allocated disk space. These contain the same first 32 bytes of the FCB, but lack the current record and random record number bytes.

filename: Name assigned to a file. A filename can include a primary filename of 1-8 characters and a filetype of 0-3 characters. A period separates the primary filename from the filetype.

file specification: Unique file identifier. A complete CP/M file specification includes a disk drive specification followed by a colon (d:), a primary filename of 1 to 8 characters, a period and a filetype of 0 to 3 characters. For example, b:example.tex is a complete CP/M file specification.

filetype: Extension to a filename. A filetype can be from 0 to 3 characters and must be separated from the primary filename by a period. A filetype can tell something about the file. Some programs require that files to be processed have specific filetypes.

floppy disk: Flexible magnetic disk used to store information. Floppy disks come in 5¼- and 8-inch diameters.

FSC: Parameter in the diskdef macro library specifying the first physical sector number. This parameter is used to determine SPT and build XLT.

hard disk: Rigid, platter-like, magnetic disk sealed in a container. A hard disk stores more information than a floppy disk.

hardware: Physical components of a computer.

hexadecimal notation: Notation for base 16 values using the decimal digits and letters A, B, C, D, E & F to represent the 16 digits. Hexadecimal notation is often used to refer to binary numbers. A binary number can be easily expressed as a hexadecimal value by taking the bits in groups of 4 starting with the least significant bit, and expressing each group as a hexadecimal digit, (0-F). Thus the bit value 1011 becomes 0BH and 10110101 becomes 0B5H.

hex file: ASCII-printable representation of a command (machine language) file.

hex file format: Absolute output of ASM and MAC for the Intel 8080 is a hex format file, containing a sequence of absolute records that give a load address and byte values to be stored, starting at the load address.

HOME: BIOS entry point which sets the disk head of the currently selected drive to the track zero position.

host: Physical characteristics of a hard disk drive in a system using the blocking and deblocking algorithm. The term "host" helps distinguish physical hardware characteris-

tics from CP/M's logical characteristics. For example, CP/M sectors are always 128 bytes, although the host sector size may be a multiple of 128 bytes.

input: Data going into the computer, usually from an operator typing at the terminal or by a program reading from the disk.

input/output: See I/O.

interface: Object that allows two independent systems to communicate with each other, as an interface between hardware and software in a microcomputer.

I/O: Abbreviation for input/output. Usually refers to input/output operations or routines handling the input and output of data in the computer system.

IOBYTE: A one byte field in page zero, currently at location 0003H, that can support a logical-to-physical device mapping for I/O. However, its implementation in your BIOS is purely optional and may or may not be supported in a given CP/M system. The IOBYTE is easily set using the command:

STAT <logical device> = <physical device>

The CP/M logical devices are CON:, RDR:, PUN:, AND LST:; each of these can be assigned to one of four physical devices. The IOBYTE may be initialized by the BOOT entry point of the BIOS and interpreted by the BIOS I/O entry points CONST, CONIN, CONOUT, LIST, PUNCH, and READER. Depending on the setting of the IOBYTE, different I/O drivers may be selected by the BIOS. For example, setting LST:=TTY: might cause LIST output to be directed to a serial port, while setting LST:=LPT: causes LIST output to be directed to a parallel port.

K: Abbreviation for kilobyte. See kilobyte.

keyword: See command keyword.

kilobyte (K): 1024 bytes or 0400H bytes of memory. This is a standard unit of memory. For example, the Intel 8080 supports up to 64K of memory address space or 65,536 bytes. 1024 kilobytes equal one megabyte, or over one million bytes.

linker: Utility program used to combine relocatable object modules into an absolute file ready for execution. For example, LINK-80 creates either a COM or PRL file from relocatable REL files, such as those produced by PL/I-80.

LIST: A BIOS entry point to a routine that sends a character to the list device, usually a printer.

list device: Device such as a printer onto which data can be listed or printed.

LISTST: BIOS entry point to a routine that returns the ready status of the list device

loader: Utility program that brings an absolute program image into memory ready for execution under the operating system, or a utility used to make such an image. For example, LOAD prepares an absolute COM file from the assembler hex file output which is ready to be executed under CP/M.

logged in: Made known to the operating system, in reference to drives. A drive is logged in when it is selected by the user or an executing process. It remains selected or logged in until you change disks in a floppy disk drive or enter ctl-C at the command level, or until a BDOS function 0 is executed.

logical: Representation of something that may or may not be the same in its actual physical form. For example, a hard disk can occupy one physical drive, yet you can divide the available storage on it to appear to the user as if it were in several different drives. These apparent drives are the logical drives.

logical sector: See **sector.**

logical to physical sector translation table: See **XLT.**

LSC: Diskdef macro library parameter specifying the last physical sector number.

LST: Logical CP/M list device (usually a printer). The CP/M list device is an output-only device referenced through the LIST and LISTST entry points of the BIOS. The STAT command allows assignment of LST: to one of the physical devices: TTY:, CRT:, LPT:, or UL1:, provided these devices and the IOBYTE are implemented in the LIST and LISTST entry points of your CP/M BIOS module. The CP/NET command NETWORK allows assignment of LST: to a list device on a network master. An example of how LST: is used in a command: PIP LST:=TEST.SUB prints the file TEST.SUB on the list device.

macro assembler: Assembler code translator providing macro processing facilities. Macro definitions allow groups of instructions to be stored and substituted in the source program as the macro names are encountered. Definitions and invocations may be nested and macro parameters can be formed to pass arbitrary strings of text to a specific macro for substitution during expansion.

megabyte: Over one million bytes; 1024 kilobytes. See **byte, kilobyte.**

microprocessor: Silicon chip that is the central processing unit (CPU) of the microcomputer. The Intel 8080 and the Zilog Z-80 are microprocessors commonly used in CP/M systems.

MOVCPM image: Memory image of the CP/M system created by MOVCPM. This image may be saved as a disk file using the SAVE command or placed on the system tracks using the SYSGEN command without specifying a source drive. This image varies, depending on the presence of a one-sector or two-sector boot. If the boot is less than 128 bytes (one sector), the boot begins at 0900H, the CP/M system at 0980H, and the BIOS at 1F80H. Otherwise, the boot is at 0900H, the CP/M system at 1000H, and the BIOS at 2000H. In a CP/M 1.4 system with a one-sector boot, the addresses are the same as for the CP/M 2 system—except that the BIOS begins at 1E80H instead of 1F80H.

MP/M: Multi-Programming Monitor control program. A microcomputer operating system supporting multi-terminal access with multi-programming at each terminal.

multi-programming: The capability of initiating and executing more than one program at a time. These programs, usually called processes, are time-shared, each receiving a slice of CPU time on a "round-robin" basis. See **concurrency.**

nibble: One half of a byte, usually the high order or low order 4 bits in a byte.

OFF: Two byte parameter in the disk parameter block at DPB + 13 bytes. This value specifies the number of reserved system tracks. The disk directory begins in the first sector of track OFF.

OFS: Diskdef macro library parameter specifying the number of reserved system tracks. See **OFF.**

operating system: Collection of programs that supervises the execution of other programs and the management of computer resources. An operating system provides an orderly input/output environment between the computer and its peripheral devices. It enables user-written programs to execute safely. An operating system standardizes the use of computer resources for the programs running under it.

option: One of many parameters that can be part of a command tail. Use options to specify additional conditions for a command's execution.

output: Data that is sent to the console, disk, or printer.

page: 256 consecutive bytes in memory beginning on a page boundary, whose base address is a multiple of 256 (100H) bytes. In hex notation, pages always begin at an address with a least significant byte of zero.

page relocatable program: See PRL.

page zero: Memory region between 0000H and 0100H used to hold critical system parameters. Page zero functions primarily as an interface region between user programs and the CP/M BDOS module. Note: in non-standard systems this region is the base page of the system and represents the first 256 bytes of memory used by the CP/M system and user programs running under it.

parameter: Value in the command tail that provides additional information for the command. Technically, a parameter is a required element of a command.

peripheral devices: Devices external to the CPU. For example, terminals, printers, and disk drives are common peripheral devices that are not part of the processor but are used in conjunction with it.

physical: Characteristic of computer components, generally hardware, that actually exist. In programs, physical components can be represented by logical components.

primary filename: First 8 characters of a filename. The primary filename is a unique name that helps the user identify the file contents. A primary filename contains 1 to 8 characters and can include any letter or number and some special characters. The primary filename follows the optional drive specification and precedes the optional filetype.

PRL: Page relocatable program. A page relocatable program is stored on diskette as a file of type PRL. Page relocatable programs are easily relocated to any page boundary and thus are suitable for execution in a non-banked MP/M system.

program: Series of coded instructions that performs specific tasks when executed by a computer. A program can be written in a processor-specific language or a high-level language that can be implemented on a number of different processors.

prompt: Any characters displayed on the video screen to help the user decide what the next appropriate action is. A system prompt is a special prompt displayed by the operating system. See CP/M prompt. The alphabetic character indicates the default drive. Some applications programs have their own special prompts.

PUN: Logical CP/M punch device. The punch device is an output-only device accessed through the PUNCH entry point of the BIOS. In certain implementations, PUN: can be a serial device such as a modem.

PUNCH: BIOS entry point to a routine that sends a character to the punch device.

RDR: Logical CP/M reader device. The reader device is an input-only device accessed through the READER entry point in the BIOS. See PUN:.

READ: Entry point in the BIOS to a routine that reads 128 bytes from the currently selected drive, track, and sector into the current DMA address.

READER: Entry point to a routine in the BIOS that reads the next character from the currently assigned reader device.

read-only (R O): Attribute that can be assigned to a disk file or a disk drive. When assigned to a file, the read-only attribute allows you to read from that file but not write to it. When assigned to a drive, the read-only attribute allows you to read any file on the disk, but prevents you from adding a new file, erasing or changing a file, renaming a file, or writing on the disk. The STAT command can set a file or a drive to read-only. Every file and drive is either read-only or read-write. The default setting for drives and files is read-write, but an error in resetting the disk or changing media automatically sets the drive to read-only until the error is corrected. See also ROM.

read-write (R W): Attribute that can be assigned to a disk file or a disk drive. The read-write attribute allows you to read from and write to a specific read-write file or to any file on a disk that is in a drive set to read-write. A file or drive can be set to either read-only or read-write.

record: Group of bytes in a file. A physical record consists of 128 bytes and is the basic unit of data transfer between the operating system and the application program. A logical record may vary in length and is used to represent a unit of information. Two 64 byte "employee" records can be stored in one 128-byte physical record. Records are grouped together to form a file.

recursive procedure: Code that may call itself during execution.

reentrant procedure: Code that can be called by one process while another is already executing it. Thus, reentrant code may be shared between different users. Reentrant procedures must not be self-modifying; that is, they must be pure code and not contain data. The data for reentrant procedures can be kept in a separate data area or placed on the stack.

restart (RST): One-byte call instruction usually used during interrupt sequences and for debugger break pointing. There are eight restart locations, RST 0 through RST 7, whose addresses are given by the product of 8 times the restart number.

RO: See read-only.

ROM: Read-only memory. This memory can be read but not written and so is suitable for code and preinitialized data areas only.

RST: See restart.

RW: See read-write.

sector: In a CP/M system, a sector is always 128 consecutive bytes. A sector is the basic unit of data read and written on the disk by the BIOS. A sector can be one 128-byte record in a file or a sector of the directory. The BDOS always requests a logical sector number between 0 and (SPT-1). This is typically translated into a physical sector by the BIOS entry point SECTRAN. In some disk subsystems, the disk sector size is larger than 128 bytes, usually a power of two such as 256, 512, 1024 or 2048 bytes. These disk sectors are

always referred to as host sectors in CP/M documentation and should not be confused with other references to sectors, in which cases the CP/M 128 byte sectors should be assumed. When the host sector size is larger than 128 bytes, host sectors must be buffered in memory and the 128 byte CP/M sectors must be blocked and deblocked from them. This may be done by adding an additional module, the blocking and deblocking algorithm, between the BIOS disk I/O routines and the actual disk I/O.

sectors per track (SPT): A two byte parameter in the disk parameter block at DPB + 0. The BDOS makes calls to the BIOS entry point SECTRAN with logical sector numbers ranging between 0 and (SPT - 1) in register BC.

SECTRAN: Entry point to a routine in the BIOS that performs logical to physical sector translation for the BDOS.

SELDSK: Entry point to a routine in the BIOS that sets the currently selected drive.

SETDMA: Entry point to a routine in the BIOS that sets the currently selected DMA address. The DMA address is the address of a 128-byte buffer region in memory that is used to transfer data to and from the disk in subsequent reads and writes.

SETSEC: Entry point to a routine in the BIOS that sets the currently selected sector.

SETTRK: Entry point to a routine in the BIOS that sets the currently selected track.

skew factor: Factor that defines the logical to physical sector number translation in XLT. Logical sector numbers are used by the BDOS and range between 0 and (SPT-1). Data is written in consecutive logical 128-byte sectors grouped in data blocks. The number of sectors per block is given by BLS/128. Physical sectors on the disk media are also numbered consecutively. If the physical sector size is also 128 bytes, a one-to-one relationship exists between logical and physical sectors. The logical to physical translation table (XLT) maps this relationship, and a skew factor is typically used in generating the table entries. For instance, if the skew factor is 6, XLT will be:

Logical:	0	1	2	3	4	5	6	. . .	25
Physical:	1	7	13	19	25	5	11	. . .	22

The skew factor allows time for program processing without missing the next sector. Otherwise, the system must wait for an entire disk revolution before reading the next logical sector. The skew factor can be varied, depending on hardware speed and application processing overhead. Note that no sector translation is done when the physical sectors are larger than 128 bytes, as sector deblocking is done in this case. (See also **sector**, **SKF** and **XLT**)

SKF: A diskdef macro library parameter specifying the skew factor to be used in building XLT. If SKF is zero, no translation table is generated and the XLT byte in the DPH will be 0000H.

software: Programs that contain machine-readable instructions, as opposed to hardware, which is the actual physical components of a computer.

source file: ASCII text file usually created with an editor, which is an input file to a system program such as a language translator or text formatter.

SP: Stack pointer. See **stack**.

spooling: Process of accumulating printer output in a file while the printer is busy. The file is printed when the printer becomes free; a program does not have to wait for the slow printing process.

SPT: See sectors per track.

stack: Reserved area of memory where the processor saves the return address when a call instruction is received. When a return instruction is encountered, the processor restores the current address on the stack to the program counter. Data such as the contents of the registers can also be saved on the stack. The push instruction places data on the stack and the pop instruction removes it. An item is pushed onto the stack by decrementing the stack pointer (SP) by 2 and writing the item at the SP address. In other words, the stack grows downward in memory.

syntax: Format for entering a given command.

SYS: See system attribute.

SYSGEN image: Memory image of the CP/M system created by SYSGEN when a destination drive is not specified. This is the same as the MOVCPM image, which can be read by SYSGEN if a source drive is not specified. See MOVCPM image.

system attribute (SYS): File attribute. You can give a file the system attribute by using the SYS option in the STAT command or by using the set file attributes function (BDOS function 12). A file with the SYS attribute is not displayed in response to a DIR command. If you give a file with user number 0 the SYS attribute, you can read and execute that file from any user number on the same drive. Use this feature to make your commonly used programs available under any user number.

system prompt: Symbol displayed by the operating system indicating that the system is ready to receive input. See prompt, CP/M prompt.

system tracks: Tracks reserved on the disk for the CP/M system. The number of system tracks is specified by the parameter OFF in the disk parameter block (DPB). The system tracks for a drive always precede its data tracks. The command SYSGEN copies the CP/M system from the system tracks to memory, and vice versa. The standard SYSGEN utility copies 26 sectors from track 0 and 26 sectors from track 1. When the system tracks contain additional sectors or tracks to be copied, a customized SYSGEN must be used.

terminal: See console.

TPA: Transient program area. Area in memory where user programs run and store data. This area is a region of memory beginning at 0100H and extending to the base of the CP/M system in high memory. The first module of the CP/M system is the CCP, which may be overwritten by a user program. If so, the TPA is extended to the base of the CP/M BDOS module. If the CCP is overwritten, the user program must terminate with either a system reset (function 0) call or a jump to location zero in page zero. The address of the base of the CP/M BDOS is stored in location 0006H in page zero, least significant byte first.

track: Data on the disk media is accessed by combination of track and sector numbers. Tracks form concentric rings on the disk; the standard IBM single-density diskettes have 77 tracks. Each track consists of a fixed number of numbered sectors. Tracks are numbered from 0 to one less than the number of tracks on the disk.

transient program area: See TPA.

upward compatible: Term meaning that a program created for the previously released operating system (or compiler, etc.) runs under the newly released version of the same operating system.

USER: Term used in CP/M and MP/M systems to distinguish distinct regions of the directory.

user number: Number assigned to files in the disk directory so that different users need only deal with their own files and have their "own" directories, even though they are all working from the same disk. In CP/M, files can be divided into 16 user groups.

utility: "Tool." Program that enables the user to perform certain operations, such as copying files, erasing files, and editing files. The utilities are created for the convenience of programmers and users.

vector: Location in memory. An entry point into the operating system used for making system calls or interrupt handling.

warm start: Program termination by: a jump to the warm start vector at location 0000H, a system reset (BDOS function 0), or a ctl-C typed at the keyboard. A warm start reinitializes the disk subsystem and returns control to the CP/M operating system at the CCP level. The warm start vector is simply a jump to the WBOOT entry point in the BIOS.

WBOOT: Entry point to a routine in the BIOS used when a warm start occurs. A warm start is performed when a user program branches to location 0000H, when the CPU is reset from the front panel, or when the user types ctl-C. The CCP and BDOS are reloaded from the system tracks of drive A.

wildcard characters: Special characters that match certain specified items. In CP/M there are two wildcard characters: ? and *. The ? can be substituted for any single character in a filename, and the * can be substituted for the primary filename or the filetype, or both. By placing wildcard characters in filenames, the user creates an ambiguous filename and can quickly reference one or more files.

word: 16-bit or two-byte value, such as an address value. Although the Intel 8080 is an 8-bit CPU, addresses occupy two bytes and are called word values.

WRITE: Entry point to a routine in the BIOS that writes the record at the currently selected DMA address to the currently selected drive, track, and sector.

XLT: Logical to physical sector translation table located in the BIOS. SECTRAN uses XLT to perform logical to physical sector number translation. XLT also refers to the two-byte address in the disk parameter header at DPBASE + 0. If this parameter is zero, no sector translation takes place. Otherwise this parameter is the address of the translation table.

ZERO PAGE: See page zero.

ALL INFORMATION PRESENTED HERE IS PROPRIETARY TO DIGITAL RESEARCH

Appendix I: CP/M Messages

Messages come from several different sources. CP/M displays error messages when there are errors in calls to the Basic Disk Operating System (BDOS). CP/M also displays messages when there are errors in command lines. Each utility supplied with CP/M has its own set of messages. The following lists CP/M messages and utility messages. One might see messages other than those listed here if one is running an application program. Check the application program's documentation for explanations of those messages.

Message — **Meaning**

?

DDT. This message has four possible meanings:
1) DDT does not understand the assembly language instruction.
2) The file cannot be opened.
3) A checksum error occurred in a HEX file.
4) The assembler/disassembler was overlayed.

ABORTED

PIP. You stopped a PIP operation by pressing a key.

ASM Error Messages

- **D** Data error: data statement element cannot be placed in specified data area.
- **E** Expression error: expression cannot be evaluated during assembly.
- **L** Label error: label cannot appear in this context (might be duplicate label).
- **N** Not implemented: unimplemented features, such as macros, are trapped.
- **O** Overflow: expression is too complex to evaluate.
- **P** Phase error: label value changes on two passes through assembly.
- **R** Register error: the value specified as a register is incompatible with the code.

S Syntax error: improperly formed expression.
U Undefined label: label used does not exist.
V Value error: improperly formed operand encountered in an expression.

BAD DELIMITER

STAT. Check command line for typing errors.

Bad Load

CCP error message, or SAVE error message.

Bdos Err On d:

Basic Disk Operating System Error on the designated drive: CP/M replaces d: with the drive specification of the drive where the error occurred. This message is followed by one of the four phrases in the situations described below.

Bdos Err On d: Bad Sector

This message appears when CP/M finds no disk in the drive, when the disk is improperly formatted, when the drive latch is open, or when power to the drive is off. Check for one of these situations and try again. This could also indicate a hardware problem or a worn or improperly formatted disk. Press ↑C to terminate the program and return to CP/M, or press the return key to ignore the error.

Bdos Err On d: File R/O

You tried to erase, rename, or set file attributes on a Read-Only file. The file should first be set to Read-Write (RW) with the command: "STAT filespec $R/W."

Bdos Err On d: R/O

Drive has been assigned Read Only status with a STAT command, or the disk in the drive has been changed without being initialized with a ↑C. CP/M terminates the current program as soon as you press any key.

Bdos Err on d: Select

CP/M received a command line specifying a nonexistent drive. CP/M terminates the current program as soon as you press any key. Press return key or CTRL-C to recover.

Break "x" at c

ED. "x" is one of the symbols described below and c is the command letter being executed when the error occurred.

Search failure. ED cannot find the string specified in an F, S, or N command.

? Unrecognized command letter c. ED does not recognize the indicated command letter, or an E, H, Q, or O command is not alone on its command line.

O The file specified in an R command cannot be found.

> Buffer full. ED cannot put any more characters in the memory buffer, or the string specified in an F, N, or S command is too long.

E Command aborted. A keystroke at the console aborted command execution.

F Disk or directory full. This error is followed by either the disk or directory full message. Refer to the recovery procedures listed under these messages.

CANNOT CLOSE DESTINATION FILE— {filespec}

PIP. An output file cannot be closed. You should take appropriate action after checking to see if the correct disk is in the drive and that the disk is not write-protected.

Cannot close, R/O
CANNOT CLOSE FILES

CP/M cannot write to the file. This usually occurs because the disk is write-protected.

ASM. An output file cannot be closed. This is a fatal error that terminates ASM execution. Check to see that the disk is in the drive, and that the disk is not write-protected.

DDT. The disk file written by a W command cannot be closed. This is a fatal error that terminates DDT execution. Check if the correct disk is in the drive and that the disk is not write-protected.

SUBMIT. This error can occur during SUBMIT file processing. Check if the correct system disk is in the A drive and that the disk is not write-protected. The SUBMIT job can be restarted after rebooting CP/M.

CANNOT READ

PIP. PIP cannot read the specified source. Reader may not be implemented.

CANNOT WRITE

PIP. The destination specified in the PIP command is illegal. You probably specified an input device as a destination.

Checksum error

PIP. A hex record checksum error was encountered. The hex record that produced the error must be corrected, probably by recreating the hex file.

CHECKSUM ERROR
LOAD ADDRESS hhhh
ERROR ADDRESS hhhh
BYTES READ:
hhhh:

LOAD. File contains incorrect data. Regenerate hex file from the source.

Command Buffer Overflow

SUBMIT. The SUBMIT buffer allows up to 2048 characters in the input file.

Command too long

SUBMIT. A command in the SUBMIT file cannot exceed 125 characters.

CORRECT ERROR, TYPE RETURN OR CTL-Z

PIP. A hex record checksum was encountered during the transfer of a hex file. The hex file with the checksum error should be corrected, probably by recreating the hex file.

DESTINATION IS R/O, DELETE (Y/N)?

PIP. The destination file specified in a PIP command already exists and it is Read Only. If you type Y, the destination file is deleted before the file copy is done.

Directory full

ED. There is not enough directory space for the file being written to the destination disk. You can use the OXfilespec command to erase any unnecessary files on the disk without leaving the editor.

SUBMIT. There is not enough directory space to write the $$$.SUB file used for processing SUBMITs. Erase some files or select a new disk and retry.

Disk full

ED. There is not enough disk space for the output file. This error can occur on the W, E, H, or X commands. If it occurs with X command, you can repeat the command prefixing the filename with a different drive.

DISK READ ERROR—{filespec}

PIP. The input disk file specified in a PIP command cannot be read properly. This is usually the result of an unexpected end-of-file. Correct the problem in your file.

DISK WRITE ERROR—{filespec}

DDT. A disk write operation cannot be successfully performed during a W command, probably due to a full disk. You should either erase some unnecessary files or get another disk with more space.

PIP. A disk write operation cannot be successfully performed during a PIP command, probably due to a full disk. You should either erase some unnecessary files or get another disk with more space and execute PIP again.

SUBMIT. The SUBMIT program cannot write the $$$.SUB file to the disk. Erase some files, or select a new disk and try again.

ERROR: BAD PARAMETER

PIP. You entered an illegal parameter in a PIP command. Retype the entry correctly.

ERROR: CANNOT OPEN SOURCE, LOAD ADDRESS hhhh

LOAD. Displayed if LOAD cannot find the specified file or if no filename is specified.

ERROR: CANNOT CLOSE FILE, LOAD ADDRESS hhhh

LOAD. Caused by an error code returned by a BDOS function call. Disk may be write-protected.

ERROR: CANNOT OPEN SOURCE, LOAD ADDRESS hhhh

LOAD. Cannot find source file. Check disk directory.

ERROR: DISK READ, LOAD ADDRESS hhhh

LOAD. Caused by an error code returned by a BDOS function call.

ERROR: DISK WRITE, LOAD ADDRESS hhhh

LOAD. Destination Disk is full.

ERROR: INVERTED LOAD ADDRESS, LOAD ADDRESS hhhh

LOAD. The address of a record was too far from the address of the previously-processed record. This is an internal limitation of LOAD, but it can be circumvented. Use DDT to read the hexfile into memory, then use a SAVE command to store the memory image file on disk.

ERROR: NO MORE DIRECTORY SPACE, LOAD ADDRESS hhhh

LOAD. Disk directory is full.

Error on line nnn message

SUBMIT. The SUBMIT program displays its messages in the format shown above, where nnn represents the line number of the SUBMIT file. Refer to the message following the line number.

FILE ERROR

ED. Disk or directory is full, and ED cannot write anything more on the disk. This is a fatal error, so make sure there is enough space on the disk to hold a second copy of the file before invoking ED.

FILE EXISTS

You have asked CP/M to create or rename a file using a file specification that is already assigned to another file. Either delete the existing file or use another file specification.

REN. The new name specified is the name of a file that already exists. You cannot rename a file with the name of an existing file. If you want to replace an existing file with a newer version of the same file, either rename or erase the existing file, or use the PIP utility.

File exists, erase it

ED. The destination filename already exists when you are placing the destination file on a different disk than the source. It should be erased or another disk selected to receive the output file.

** FILE IS READ/ONLY **

ED. The file specified in the command to invoke ED has the Read Only attribute. ED can read the file so that the user can examine it, but ED cannot change a Read Only file.

File Not Found

CP/M cannot find the specified file. Check that you have entered the correct drive specification or that you have the correct disk in the drive.

ED. ED cannot find the specified file. Check that you have entered the correct drive specification or that you have the correct disk in the drive.

STAT. STAT cannot find the specified file. The message might appear if you omit the drive specification. Check to see if the correct disk is in the drive.

FILE NOT FOUND—{filespec}

PIP. An input file that you have specified does not exist.

Filename required

ED. You typed the ED command without a filename. Reenter the ED command followed by the name of the file you want to edit or create.

hhhh??=dd

DDT. The ?? indicates DDT does not know how to represent the hexadecimal value dd encountered at address hhhh in 8080 assembly language. dd is not an 8080 machine instruction opcode.

Insufficient memory

DDT. There is not enough memory to load the file specified in an R or E command.

Invalid Assignment

STAT. You specified an invalid drive or file assignment, or misspelled a device name. This error message might be followed by a list of the valid file assignments that can follow a filename. If an invalid drive assignment was attempted the message "Use: d:=RO" is displayed, showing the proper syntax for drive assignments.

Invalid control character

SUBMIT. The only valid control characters in the SUBMIT files of type SUB are ^A through ^Z. Note that in a SUBMIT file the control character is represented by typing the circumflex, ^, not by pressing the control key.

INVALID DIGIT—{filespec}

PIP. An invalid hex digit has been encountered while reading a hex file. The hex file with the invalid hex digit should be corrected, probably by recreating the hex file.

Invalid Disk Assignment

STAT. Might appear if you follow the drive specification with anything except =R/O.

INVALID DISK SELECT

CP/M received a command line specifying a nonexistent drive, or the disk in the drive is improperly formatted. CP/M terminates the current program as soon as you press any key.

INVALID DRIVE NAME (Use A, B, C, or D)

SYSGEN. SYSGEN recognizes only drives A, B, C and D as valid destinations for system generation.

Invalid File Indicator

STAT. Appears if you do not specify RO, RW, DIR, or SYS.

INVALID FORMAT

PIP. The format of your PIP command is illegal. See the description of the PIP command.

INVALID HEX DIGIT
LOAD ADDRESS hhhh
ERROR ADDRESS hhhh
BYTES READ:
hhhh

LOAD. File contains incorrect hex digit.

INVALID MEMORY SIZE

MOVCPM. Specify a value less than 64K or your computer's actual memory size.

INVALID SEPARATOR

PIP. You have placed an invalid character for a separator between two input filenames.

INVALID USER NUMBER

PIP. You have specified a user number greater than 15. User numbers are in the range 0 to 15.

n?

USER. You specified a number greater than fifteen for a user area number. For example, if you type USER 18<cr>, the screen displays 18?.

NO DIRECTORY SPACE

ASM. The disk directory is full. Erase some files to make room for PRN and HEX files. The directory can usually hold only 64 filenames.

NO DIRECTORY SPACE—{filespec}

PIP. There is not enough directory space for the output file. You should either erase some unnecessary files or get another disk with more directory space and execute PIP again.

NO FILE—{filespec}

DIR, ERA, REN, PIP. CP/M cannot find the specified file, or no files exist.

ASM. The indicated source or include file cannot be found on the indicated drive.

DDT. The file specified in an R or E command cannot be found on the disk.

NO INPUT FILE PRESENT ON DISK

DUMP. The file you requested does not exist.

No memory

There is not enough (buffer?) memory available for loading the program specified.

NO SOURCE FILE ON DISK

SYSGEN. SYSGEN cannot find CP/M either in CPMxx.com form or on the system tracks of the source disk.

NO SOURCE FILE PRESENT

ASM. The assembler cannot find the file you specified. Either you mistyped the filespecification in your command line, or the file is not type ASM.

NO SPACE

SAVE. Too many files are already on the disk, or no room is left on the disk to save the information.

No SUB file present

SUBMIT. For SUBMIT to operate properly, you must create a file with filetype of SUB. The SUB file contains usual CP/M commands. Use one command per line.

NOT A CHARACTER SOURCE

PIP. The source specified in your PIP command is illegal. You have probably specified an output device as a source.

** NOT DELETED **

PIP. PIP did not delete the file, which may have had the R/O attribute.

NOT FOUND

PIP. PIP cannot find the specified file.

OUTPUT FILE WRITE ERROR

ASM. You specified a write-protected diskette as the destination for the PRN and HEX files, or the diskette has no space left. Correct the problem before assembling your program.

Parameter error

SUBMIT. Within the SUBMIT file of type sub, valid parameters are $0 through $9.

PARAMETER ERROR, TYPE RETURN TO IGNORE

SYSGEN. If you press return, SYSGEN proceeds without processing the invalid parameter.

QUIT NOT FOUND

PIP. The string argument to a Q parameter was not found in your input file.

Read error

TYPE. An error occurred when reading the file specified in the type command. Check the disk and try again. The STAT filespec command can diagnose trouble.

READER STOPPING

PIP. Reader operation interrupted.

Record Too Long

PIP. PIP cannot process a record longer than 128 bytes.

Requires CP/M 2.0 or later

XSUB. XSUB requires the facilities of CP/M 2.0 or newer version.

Requires CP/M 2.0 or newer for operation

PIP. This version of PIP requires the facilities of CP/M 2.0 or newer version.

START NOT FOUND

PIP. The string argument to an S parameter cannot be found in the source file.

SOURCE FILE INCOMPLETE

SYSGEN. SYSGEN cannot use your CP/M source file.

SOURCE FILE NAME ERROR

ASM. When you assemble a file, you cannot use the wildcard characters * and ? in the filename. Only one file can be assembled at a time.

SOURCE FILE READ ERROR

ASM. The assembler cannot understand the information in the file

containing the assembly language program. Portions of another file might have been written over your assembly language file, or information was not properly saved on the diskette. Use the TYPE command to locate the error. Assembly language files contain the letters, symbols, and numbers that appear on your keyboard. If your screen displays unrecognizable output or behaves strangely, you have found where computer instructions have crept into your file.

SYNCHRONIZATION ERROR

MOVCPM. The MOVCPM utility is being used with the wrong CP/M system.

"SYSTEM" FILE NOT ACCESSIBLE

You tried to access a file set to SYS with the STAT command.

** TOO MANY FILES **

STAT. There is not enough memory for STAT to sort the files specified, or more than 512 files were specified.

UNEXPECTED END OF HEX FILE—{filespec}

PIP. An end-of-file was encountered prior to a termination hex record. The hex file without a termination record should be corrected, probably by recreating the hex file.

Unrecognized Destination

PIP. Check command line for valid destination.

Use: STAT d:=RO

STAT. An invalid STAT drive command was given. The only valid drive assignment in STAT is STAT d:=RO.

VERIFY ERROR:—{filespec}

PIP. When copying with the V option, PIP found a difference when rereading the data just written and comparing it to the data in its memory buffer. Usually this indicates a failure of either the destination disk or drive.

WRONG CP/M VERSION (REQUIRES 2.0)

XSUB ACTIVE

SUBMIT. XSUB has been invoked.

XSUB ALREADY PRESENT

SUBMIT. XSUB is already active in memory.

Your input?

If CP/M cannot find the command you specified, it returns the command name you entered followed by a question mark. Check that you have typed the command line correctly, or that the command you requested exists as a .COM file on the default or specified disk.

INDEX

Absolute line number, 36
Access mode, 13
afn (ambiguous file reference), 3, 4, 6
Allocation vector, 105
Ambiguous file reference (afn), 3, 4, 6
ASM, 15, 47
Assembler, 15, 47
Assembler/disassembler module (DDT), 77
Assembly errors, 62
Assembly language mnemonics in DDT, 71, 74
Assembly language program, 49
Assembly language statement, 49
Automatic command processing, 25

Base, 50
Basic Disk Operating System (BDOS), 2, 89, 127
Basic I/O System (BIOS), 2, 89, 127
BDOS (Basic Disk Operating System), 2, 89, 127
Binary constants, 50
BIOS (Basic I/O System), 2, 89, 127
BIOS disk definition, 148
BIOS subroutines, 137
Block move command, 74
bls parameter, 149
BOOT, 90, 136, 140
BOOT entry point, 140
Breakpoint, 71, 73
Built-in commands, 3

Case translation, 5, 6, 20, 21, 37, 39, 44, 45, 51, 95
CCP (Console Command Processor), 2, 69, 89, 127
CCP Stack, 92
Character pointer, 35
CKS parameter, 149
Close File function, 101
Code and data areas, 144
Cold start loader, 136, 140, 143
Combine files, 17
Command, 3
Command line, 90
Comment field, 49
Compute File Size function, 108
Condition flags, 58, 77
Conditional assembly, 56
CONIN, 140
CONOUT, 141
CONSOLE, 138
Console Command Processor (CCP), 2,69, 89, 127
Console Input function, 95
Console Output function, 96
CONST, 140
Constant, 50
Control characters, 44
Control functions, 9

Control-Z character, 93
Copy files, 17
CPU state, 71
cr (carriage return), 39
Create files, 23
Create system disk, 24
Creating COM files, 16
Currently logged disk, 3, 5, 10, 17, 25

Data allocation size, 147
Data block number, 147
DB statement, 57
DDT commands, 70, 133
DDT nucleus, 77
DDT prompt, 70
DDT sign-on message, 69
Decimal constant, 50
Default FCB, 73
Delete File function, 102
DESPOOL, 138
Device assignment, 11
DIR, 6
DIR attribute, 14
dir parameter, 149
Direct console I/O function, 97
Direct Memory Address, 104
Directory, 6
Directory code, 100, 101, 102, 103
Disassembler, 71, 77
Disk attributes, 11
Disk drive name, 5
Disk I/O functions, 99-110
Disk parameter block, 146
Disk parameter header, 145
Disk parameter table, 145
Disk statistics, 10
Disk-to-disk copy, 18
DISKDEF macro, 149
Diskette format, 31
DISKS macro, 150, 186
Display file contents, 8
dks parameter, 149
DMA, 104
DMA address, 93
dn parameter, 149
DPBASE, 146
Drive characteristics, 14
Drive select code, 94
Drive specification, 5
DS statement, 57
DUMP, 27, 113
DW statement, 57

ED, 23, 33-45, 131
ED commands, 38, 44
ED errors, 43
Edit command line, 9
8080 CPU registers, 76
8080 registers, 51
end-of-file, 19, 93
END statement, 49, 54
ENDEF macro, 150
ENDIF statement, 56
EQU statement, 55

ERA, 6
Erase files, 6
Error messages, 29, 43, 62, 153
Expression, 49
Extents, 13

FBASE, 89
FCB, 93, 94
FCB format, 93, 94
FDOS (operations), 89, 91
File attributes, 14
File compatibility, 23
File control block (FCB), 93, 94
File expansion, 128
File extent, 93
File indicators, 14
File names, 3
Fiel reference, 3
File statistics, 10, 13
Filetype, 93
Find command, 39
fsc parameter, 149

Get ADDR (Alloc) function, 105
Get ADDR (Disk Parms) function, 106
Get Console Status, 99
Get I/O Byte function, 97
Get Read/Only Vector function, 105
GETSYS, 128, 134

Hexadecimal constant, 50
Hex files, 16, 19, 20, 47
HOME subroutine, 139, 141

Identifier, 49, 50
IF statement, 56
Initialized storage areas, 57
In-line assembly language, 71
Insert mode, 37
Insert string, 40
IOBYTE function, 138 ,139

Jump vector, 137
Juxtaposition command, 41

Key fields, 109

Label field, 49
Labels, 48, 49, 58
Library read command, 42
Line-editing control characters, 38, 70, 98
Line-editing functions, 9
Line numbers, 36
LIST, 138, 141
List Output function, 96
LISTST, 142
LOAD, 16
Logged in, 3
Logical devices, 11, 18, 138
Logical extents, 93
Logical-physical assignments, 12, 139
Logical to physical device mapping, 138
Logical to physical sector translation, 143, 149
lsc parameter, 149

Machine executable code, 16
Macro command, 42
Make File function, 103
Memory buffer, 33, 34, 35, 37
Memory image, 71, 131, 132
Memory image file, 16
Memory size, 27, 128, 132
MOVCPM, 27, 131, 132
Multiple command processing, 25

Negative bias, 132

[o] parameter, 149
Octal constant, 50
ofs parameter, 150
On-line status, 100
Open File function, 100
Operand field, 49-51
Operation field, 49-58
Operators, 52, 53, 58
ORG directive, 54

Page zero, 144
Patching the CP/M system, 128
Peripheral devices, 138
Physical devices, 12, 18, 139
Physical file size, 109
Physical to logical device assignment, 12, 139
PIP, 17
PIP devices, 19
PIP parameters, 20
Print String function, 98
PRN file, 47
Program counter, 71, 73, 76
Program tracing, 75
Prompt, 3
Pseudo-operation, 53
PUNCH, 138, 141
Punch Output function, 96
PUTSYS, 129, 135

Radix indicators, 50
Random access, 107, 108, 117
Random access files, 93
Random record number, 108
READ, 142
Read Console Buffer function, 98
Read only, 14
Read/only status, 14
Read random error codes, 107
Read Random function, 107
READ routine, 139
Read Sequential function, 102
Read/write, 14
READER, 138, 141
Reader Input function, 96
REN, 7
Rename file function, 104
Reset Disk function, 99
Reset Drive function, 109
Reset state, 99
Return Current Disk function, 104
Return Log-in Vector function, 104
Return Version Number function, 99
R/O, 14

R/O attribute, 106
R/O bit, 105
R/W, 14

SAVE, 7
SAVE command, 70
Search for First function, 101
Search for Next function, 102
Search strings, 39
Sector allocation, 136
SECTRAN, 143
SELDSK, 139, 141, 146
Select Disk function, 100
Sequential access, 93
Set DMA address function, 104
Set File Attributes function, 106
Set/Get User Code function, 106
Set I/O Byte function, 97
Set Random Record function, 109
SET statement, 55
SETDMA, 142
SETSEC, 142
SETTRK, 141
Simple character I/O, 138
Size in records, 13
skf parameter, 149, 150
Source files, 93
Stack pointer, 92
STAT, 10, 139, 151
Stop console output, 9
String substitutions, 40
SUBMIT, 25
SYS attribute, 14
SYSGEN, 24, 134
System attribute, 44, 106
System parameters, 140
System (re)initialization, 138
System Reset function, 95

Testing and debugging of programs, 69
Text transfer commands, 35
TPA (Transient Program Area), 2, 89
Trace mode, 76
Transient commands, 3, 9
Transient Program Area (TPA), 2, 89
Translate table, 150
Translation vectors, 146
TYPE, 8

ufn, 3, 6
Unambiguous file reference, 3, 6
Uninitialized memory, 57
Untrace mode, 76
USER, 8
USER numbers, 8, 15, 106

Verify line numbers command, 37, 45
Version independent programming, 99
Virtual file size, 108

Warm start, 90, 140
WBOOT entry point, 140
WRITE, 142
Write Protect Disk function, 105
Write random error codes, 108

Write Random function, 108
Write Random with Zero Fill function, 110
WRITE routine, 142
Write Sequential function, 103

XSOB, 26

APPENDIX C

CUSTOMER INFORMATION

APPENDIX C-1

SERVICING AND WARRANTY INFORMATION

SERVICING PROCEDURES

Your CompuStar product is warranted to the original purchaser for 90 days from date of shipment. This warranty covers the adjustment or replacement, F.O.B. Intertec's plant in Columbia, South Carolina, of any part or parts which in Intertec's judgment shall disclose to have been originally defective. A complete statement of your warranty rights is contained on the inside back cover of this manual.

To qualify for receipt of future technical documentation updates, please complete the Warranty Registration Form (contained in this section) and return it to Intertec Data Systems within 10 days of receipt of this equipment. Be sure to include the serial number of the specific terminal you are registering. This serial number can be found underneath and toward the back of your terminal. A Customer Comment Card is also enclosed for your convenience if you desire to make comments regarding the overall operation and/or adaptability of the CompuStar to your particular application.

IF SERVICE IS EVER REQUIRED:

If you should ever encounter difficulties with the use or operation of this terminal, contact the supplier from whom the unit was purchased for instructions regarding the proper servicing techniques. Service procedures differ from dealer to dealer, but most Intertec authorized service dealers can provide local, on-site servicing of this equipment on a per-call or maintenance contract basis. Plus, a variety of service programs are available directly from the factory, including extended warranty, a module exchange program, and on-site maintenance from a wide variety of locations within the U.S.

Contact our Customer Services Department at the factory for rates and availability if you desire to participate in one of these programs. **If you are not covered under one of the programs described above and service cannot be made available through your local supplier, contact Intertec's Customer Services Department at (803) 798-9100. Be prepared to give the following information when you call:**

1. The serial number of the defective equipment. If you are returning individual modules to the factory for repair, it will be necessary to have the serial number of the individual modules also. The serial number may be found at the back underneath your terminal. Module serial numbers are listed on white stickers placed in conspicuous locations on each major module or subassembly of the terminal. **NOTE:** Individual modules **cannot** be returned to the factory for repair unless you originally purchased your unit from the factory. If your unit was purchased through a Dealer or OEM vendor, and you desire factory repair, then the entire terminal must be returned.

2. The name and location of the Dealer and/or Agent from whom the unit was purchased.

3. A complete description of the alleged failure (including the nature and cause of the failure if readily available).

The Customer Services Department will issue you a Return Material Authorization Number (RMA Number) which will be valid for a period of 30 days. This RMA Number will be your official authorization to return equipment to IDSC for repair only. The Customer Services Department will also give you an estimate, if requested, of the time it should take to process and repair your equipment. Turnaround time on repairs varies depending on workloads and availability of parts, but normally your equipment will be repaired and returned to you within 10 working days of its receipt. If your repair is urgent, you may authorize a special $50 Emergency Repair fee and have your equipment repaired and returned within no more than 48 hours of its receipt at our Service Center. Ask the Customer Services Department for more information about this program.

IMPORTANT: Any equipment returned to Intertec without an RMA Number will result in the equipment being refused and possible cancellation of your CompuStar warranty. Also if your RMA Number expires, you must request a new number. Equipment arriving at Intertec bearing an expired RMA Number will also be refused.

After securing an RMA Number from the Customer Services Department, return the specified modules and/or complete terminals to Intertec, freight prepaid, at the address below. **NOTE: The RMA Number must be plainly marked and visible on your shipping label to prevent the equipment from being refused at Intertec's Receiving Department.**

ATTN: COMPUSTAR SERVICE CENTER
Intertec Data Systems Corporation
2300 Broad River Road
Columbia, South Carolina 29210

To aid our technicians in troubleshooting and correcting your reported malfunction, please complete an Intertec Equipment Malfunction Report (contained in this section) and enclose it with the equipment you intend to return to the factory.

Be sure a declared value equal to the price of the unit is shown on the Bill of Lading, Express Receipt or Air Freight Bill, whichever is applicable. Risk of loss or damage to equipment during the time it is in transit either to or from Intertec's facilities is your sole responsibility. A declared value must be placed on your Bill of Lading to insure substantiation of your freight claim if shipping damage or loss is incurred.

All equipment returned to an Intertec Service Center must be freight prepaid. Equipment not prepaid on arrival at Intertec's Receiving Department cannot be accepted. Upon repair of equipment under warranty, it will be returned to you freight prepaid, via UPS or equivalent ground transportation. All repaired equipment not covered by warranty will be returned, F.O.B. the factory in Columbia, South Carolina, via UPS or equivalent ground transportation unless you specify otherwise.

INSTRUCTIONS FOR HANDLING LOST OR DAMAGED EQUIPMENT

The goods described on your Packing Slip were delivered to the Transportation Company at Intertec's premises in complete and good condition. If any of the goods called for on this Packing Slip are short or damaged, you must file a claim WITH THE TRANSPORTATION COMPANY FOR THE AMOUNT OF THE DAMAGE AND/OR LOSS.

IF LOSS OR DAMAGE IS EVIDENT AT TIME OF DELIVERY:

If any of the goods called for on your Packing Slip are short or damaged at the time of delivery, ACCEPT THEM, **but insist that the Freight Agent make a damaged or short notation on your Freight Bill or Express Receipt and sign it.**

IF DAMAGE OR LOSS IS CONCEALED AND DISCOVERED AT A LATER DATE:

If any concealed loss or damage is discovered, notify your local Freight Agent or Express Agent AT ONCE and request him to make an inspection. This is absolutely necessary. Unless you do this, the Transportation Company will not consider your claim for loss or damage valid. If the agent refuses to make an inspection, you should draw up an affidavit to the effect that you notified him on a certain date and that he failed to make the necessary inspection.

After you have ascertained the extent of the loss or damage, **ORDER THE REPLACEMENT PARTS OR COMPLETE NEW UNITS FROM THE FACTORY.** We will ship them to you **and bill you for the**

cost. This new invoice will then be a part of your claim for reimbursement from the Transportation Company. This, together with other papers, will properly support your claim.

IMPORTANT: The claims adjustment procedure for UPS shipments varies somewhat from the procedure listed above for regular motor and air freight shipments. *If your equipment was shipped via UPS and sustained either damage or loss, the UPS representative in your area* **must** *initiate the claim by inspecting the goods and assigning a freight claim number to the damaged equipment.* The representative will attach a "Call Tag" to the outside of the equipment box which will be your authorization to return the merchandise to our factory for claim adjustment. Upon receipt of this damaged equipment, we will perform the necessary repairs, process the appropriate paperwork with UPS and return the equipment to you. Please allow time for processing of any type claim. Normal time for proper processing of a UPS claim is 15-30 working days.

Remember, it is extremely important that you **do not give the Transportation Company a clear receipt if damage or shortages are evident upon delivery.** It is equally important that you call for an inspection if the loss or damage is discovered later. **DO NOT, UNDER ANY CIRCUMSTANCES, ORDER THE TRANSPORTATION COMPANY TO RETURN SHIPMENT TO OUR FACTORY OR REFUSE SHIPMENT UNLESS WE HAVE AUTHORIZED SUCH RETURN.**

ADDITIONAL TECHNICAL DOCUMENTATION

Detailed technical documentation (i.e., schematics) describing the operation of the CompuStar product and the electrical interconnection of its various modules is available at nominal cost directly from Intertec Data Systems Corporation. However, due to the confidentiality of this technical information, it will be necessary to sign and return the Documentation Non-Disclosure Agreement (appearing on the next page) denoting your concurrence with its terms and conditions.

The handling and processing costs of CompuStar technical documentation is $50. Due to the large amount of requests being processed and the relatively small handling costs involved, *we* **must** *request that you* **enclose payment** *($50) upon return of your Non-Disclosure Agreement.* Normally the documents will be mailed to you within 15 to 30 days after receipt of your payment and a signed copy of the Agreement. (IMPORTANT: The technical documentation will be mailed to the address listed at the top of the Non-Disclosure Agreement.) For prompt processing of your documentation request, please forward your signed agreement and payment to:

Customer Services Department
Intertec Data Systems Corporation
2300 Broad River Road
Columbia, South Carolina 29210

NOTE: Formal technical documentation for the CompuStar will be sent to you normally within 10-15 days of receipt of your payment and signed Non-Disclosure Agreement.

IMPORTANT: *Payment* **must** *accompany your Non-Disclosure Agreement. Agreements sent to us without payment will be discarded without notice.*

APPENDIX C-2

GENERAL INFORMATION FOR COMPUSTAR USERS

INTERTEC DATA SYSTEMS®

COMPUSTAR DOCUMENTATION NON-DISCLOSURE AGREEMENT

IDS-350B

Corporate Headquarters: 2300 Broad River Road, Columbia, South Carolina 29210 • 803/798-9100 • TWX: 810-666-2115

THIS AGREEMENT MADE BETWEEN INTERTEC DATA SYSTEMS CORPORATION AND THE ORGANIZATION AND/OR PERSONS LISTED AT THE RIGHT AND BECOMES EFFECTIVE ON THE DATE SPECIFIED BELOW.

(PLEASE PRINT CLEARLY. DOCUMENTS WILL BE MAILED TO THE ADDRESS AT RIGHT)

YOUR COMPANY NAME _____

ADDRESS _____

CITY & STATE _____

TELEPHONE _____

YOUR NAME _____

For and in consideration of receiving confidential documentation on the CompuStar™ line of terminals manufactured by INTERTEC DATA SYSTEMS CORPORATION (hereinafter called INTERTEC) at the date hereof, the undersigned hereby agrees with INTERTEC as follows:

(1) The undersigned acknowleges that formulae, programs, manufacturing processes, devices, techniques, plans, methods, drawings, blueprints, reproductions, data tables, calculations and components were designed and developed by INTERTEC at great expense and over lengthy periods of time, and the same are secret and confidential, are unique and constitute the exclusive property and trade secrets of INTERTEC, and that any use of such property and trade secrets by the undersigned other than for the sole benefit of INTERTEC would be wrongful, tortiuous and would cause irreparable injury to INTERTEC.

(2) The undersigned shall not at any time, without the express written consent of the Board of Directors of INTERTEC, publish, disclose, use or divulge to any person, firm or corporation, directly or indirectly, or use for his own benefit or the benefit of any person, firm, or use other than to effect repair of INTERTEC manufacturing equipment, and property above described, trade secrets or confidential information of INTERTEC, its subsidiaries and its affiliates learned or obtained by its subsidiaries and its affiliates learned or obtained by him from INTERTEC, including, but not limited to the information and things set forth in paragraph 1 hereinabove.

(3) This agreement shall be binding upon the undersigned, his personal representatives, successors and assigns, and shall run to the benefit of INTERTEC, its successors and assigns.

(4) Upon termination of the association of the undersigned with INTERTEC or its subsidiaries, the undersigned shall promptly deliver to INTERTEC all drawings, blueprints, reproductions, manuals, letters, notes, notebooks, reports, data, tables, calculations or copies thereof, components, programs, and any and all other secret and confidential property of INTERTEC, its subsidiaries and affiliates, including, but not limited to, all of the property set forth in paragraph 1 hereinabove which are in the possession or under the control of the undersigned.

(5) The undersigned hereby acknowledges and agrees that in the event of any violation hereof, INTERTEC shall be authorized and entitled to obtain from any court of competent jurisdiction preliminary and permanent injunctive relief as well as equitable accounting of all profits or benefits arising out of such violation which rights or remedies shall be cumulative and in addition to any rights or remedies to which INTERTEC may be entitled and that the undersigned shall further be directly liable for any and all reasonable attorney's fees incurred by INTERTEC to enforce this Agreement against the undersigned in a court of law.

(6) The foregoing understanding shall apply to any subsequent meeting and/or communications between INTERTEC and the above mentioned organization relating to the same subject manner, unless modified in writing as to any such subsequent meetings and/or communications.

We would appreciate your signing and returning to us, prior to the release of INTERTEC product documentation, the original copy of this agreement denoting your concurrence with the foregoing provisions.

AGREED TO: _____

(YOUR NAME OR COMPANY - PLEASE PRINT)

YOUR SIGNATURE: _____

In addition to the terms listed above, I further certify that I am duly authorized to sign this document on behalf of the organization and/or persons requesting that this imformation be supplied by INTERTEC.

YOUR NAME: _____

YOUR TITLE: _____

TODAY'S DATE: _____

INTERTEC DATA SYSTEMS CORPORATION

SIGNATURE: _____

FOR OFFICE USE ONLY

DATE RCV'D _____ PROCESSED BY: _____

OTHER RELEASES	DATE	INVOICE NO.

INTERTEC DATA SYSTEMS®
Corporate Headquarters: 2300 Broad River Road, Columbia, South Carolina 29210 • 803/798-9100 • TWX: 810-666-2115

BE SURE TO INCLUDE YOUR SERIAL NUMBER HERE.
SERIAL NO. _____

COMPUSTAR LIMITED WARRANTY REGISTRATION FORM

IMPORTANT: This form should be completed within ten days of receipt of your CompuStar Video Processing Unit and returned to Intertec at the following address:

Intertec Data Systems Corporation
2300 Broad River Road
Columbia, South Carolina 29210

Attn: Warranty Registration Department

All warranty liability is limited to that expressed in most recent edition of the CompuStar User's Manual as published by Intertec Data Systems Corporation.

****

Date Received: _____ Purchased from: _____

Company: _____ _____

Name: _____ Address _____

Title: _____ City: _____

Address _____ Telephone: ( ) _____

City: _____ Sales Agent: _____

Country: _____ Order Placed On: _____

Telephone: ( ) _____ Price Paid: _____

****

Where did you first hear about the CompuStar? From a ☐ Magazine ☐ Dealer ☐ Friend

_____

Why did you decide to purchase the CompuStar? ☐ Features ☐ Price ☐Appearance

_____

_____

Was the Dealer and/or Sales Agent knowlegeable about the CompuStar? ☐YES ☐NO

Please explain.

_____

_____

_____

Questions on the reverse side must be completed to validate your warranty

Were you introduced to any other Intertec products? ☐ YES ☐NO (if yes, please indicate other products which were mentioned.)_____

_____

_____

_____

Are you aware of other Intertec products? ☐ YES ☐ NO (If yes, which ones?)

_____

_____

_____

What other microcomputer related products will you be purchasing in the next 12 months?

☐ Video Terminals ☐ Printers (matrix) ☐ Printers (character) ☐ Disk Systems ☐ Other

_____

_____

What is your application for the CompuStar? ☐ Business ☐ Scientific ☐ Educational ☐ Other

_____

_____

_____

_____

What are your comments in general concerning the overall operation of the CompuStar?

☐ Outstanding ☐ Excellent ☐ Good ☐ Average ☐ Unsatisfactory

_____

_____

_____

Would you like to be placed on our mailing lists? ☐ YES ☐ NO

May we use your name as a favorable reference for other customers in your area desiring to purchase a CompuStar? ☐ YES ☐ NO

Thank you for purchasing the CompuStar Video Processing Unit. If we may be of further assistance to you, please contact our Customer Service Department at the address on the reverse side of this form.

INTERTEC DATA SYSTEMS®
Corporate Headquarters: 2300 Broad River Road, Columbia, South Carolina 29210 • 803/798-9100 • TWX: 810-666-2115

EQUIPMENT MALFUNCTION REPORT

IDS - 505A

THIS EQUIPMENT PURCHASED FROM: _____

Dealer Name: _____ Address _____

City & State _____ Telephone Area (_____) _____

Dear Customer:

We are trying to manufacture the most reliable product possible. You would do us a great courtesy by completing this form should you experience any failures. Enclose this form with the equipment you intend to return to the dealer or factory for service. (Additional copies of this form available upon request.)

1. Type Unit _____ Serial No. _____

   Module (if applicable) _____

2. Component failed (if available, include Name and Number) _____

3. Description of failure (include cause of failure if readily available) _____

   _____

   _____

   _____

4. Approximate hours/days of operation to failure _____

5. Failure occurred during:

   ☐ Initial Inspection ☐ Customer Installation ☐ Field Use

6. Personal Comment:

   _____

   _____

   _____

Your Name _____ Address _____

City & State _____ Zip _____ Phone (_____) _____

Date _____ Signed _____

Return this form and equipment to your local dealer or to the factory at the address below.

ATTN: COMPUSTAR SERVICE CENTER
Intertec Data Systems Corporation
2300 Broad River Road
Columbia, South Carolina 29210

INTERTEC DATA SYSTEMS®
Corporate Headquarters: 2300 Broad River Road, Columbia, South Carolina 29210 • 803/798-9100 • TWX: 810-666-2115

CUSTOMER COMMENT CARD

Our past and present customers are directly responsible for the evolution of the CompuStar as you see it presented in this manual. Before Intertec began research and development on the CompuStar, an extensive user survey was conducted to ascertain optimum video computer price/performance ratios to enable us to capture a major portion of the video computer market. In order that we continue with our commitment to excellence in engineering, production and marketing, we would appreciate your comments below regarding your overall opinion of the CompuStar. All comments are given careful consideration in future product design and become the property of Intertec Data Corporation.

(1) What are your comments concerning the overall appearance of the CompuStar? (You may want to comment on color, size and construction.)

(2) What are your comments (in general) concerning the overall operation of the unit?

(3) What features about the unit do you like best?

(4) What features about the unit do you like least?

Please see additional questions on the reverse side

(5) Briefly describe your application for the CompuStar.

_____

_____

_____

_____

(6) What other microcomputer systems do you feel are comparable to the CompuStar in both price and performance?

_____

_____

_____

_____

(7) What changes and/or modifications to the CompuStar could be made to render it more suited to your application?

_____

_____

_____

_____

(8) Your candid comments regarding the operation of and application for the CompuStar are greatly appreciated. Address your comments and/or suggestions to:

PRODUCT SERVICES MANAGER
Intertec Data Systems
2300 Broad River Road
Columbia, South Carolina 29210

(9) If you desire to be contacted by our service, marketing or technical staff regarding these comments, please give us your complete name, address and phone number below. (This information is optional.)

Company Name_____

Address _____

City, State & Zip_____

Contact: _____

Phone: AREA (_____) _____ EXT _____

I would like to be contacted by your: ☐ Marketing ☐ Technical ☐ Service Department

INTERTEC DATA SYSTEMS
Corporate Headquarters: 2300 Broad River Road, Columbia, South Carolina 29210 • 803/798-9100 • TWX: 810-666-2115

DOCUMENT REVIEW FORM

In order to insure that you are provided with a document that will satisfy all of your information requirements as well as one that is error free and easy to use, we would like to ask you to supply us with any comments, suggestions, or errors you have found. The space below is provided for this input. Return the completed form to:

ATTN: TECHNICAL SERVICES MANAGER
Intertec Data Systems Corporation
2300 Broad River Road
Columbia, South Carolina 29210

STATEMENT OF LIMITED WARRANTY

For ninety (90) days from the date of shipment from our manufacturing plant at 2300 Broad River Road, Columbia, South Carolina, Intertec warrants, to the original purchaser only, that its products, excluding software products, will be free of defective parts or components and agrees to replace or repair any defective component which, in Intertec's judgement, shall disclose to have been originally defective. Intertec neither offers nor implies any warranty whatsoever on any software products. Furthermore, Intertec's obligations under this limited warranty are subject to the following conditions:

LIMITED WARRANTY REPAIRS

Unless authorized by written statement from Intertec, all repairs must be done by Intertec at our plant in Columbia, South Carolina. Return of any and all parts and/or equipment must be freight prepaid and accompanied by an Intertec Return Material Authorization number which must be clearly visible on the customer's shipping label. Return of parts or equipment contrary to this policy shall result in the material being refused, and the customer being invoiced for any replacement parts, if any were previously issued, at Intertec's standard prices.

When making repairs or replacing parts in accordance with this limited warranty, Intertec reserves the right to alter and/or modify specifications of this equipment.

Upon completion of the repairs, Intertec will return the equipment, freight prepaid, directly to the customer from whom it was sent via UPS or equivalent ground transportation.

Authorization to return equipment for repair can be obtained by writing Intertec at the address stated herein or by calling our Customer Service Department at 803/798-9100.

In the event Intertec shall authorize repair of its equipment, in writing, by an authorized repair agent, then Customer shall bear all shipping, packing, inspection and insurance costs necessary to effectuate repairs under this warranty.

EXCLUSIONS

The Limited Warranty provided by Intertec Data Systems Corporation does hot include:

(a) Any damage or defect caused by injuries received in shipment or any damage caused by unauthorized repairs or adjustments. The risk of loss or damage to the equipment shall pass to the Customer upon delivery by Intertec to the carrier at Intertec's premises.

(b) Repair, damage or increase in service time caused by failure to continually provide a suitable installation environment including, but not limited to, the failure to provide, or the failure of, adequate electrical power, air-conditioning, or humidity control.

(c) Repair, damage or increase in service time caused by accident or disaster, which shall include, but not be limited to, fire, flood, water, wind, lightning, transportation neglect, misuse and alterations, which shall include, but not be limited to, any deviation from the original physical, mechanical or electrical design of the product.

(d) Any statements made about the equipment by salesman, dealers or agents unless such statements are in a written document signed by an officer of Intertec Data Systems Corporation. Such statements do not constitute warranties, shall not be relied on by the buyer, and are not part of the contract for sale.

(e) Any damage arising out of any application for its products other than for normal commercial and industrial use, unless such application is, upon request, specifically approved in writing by Intertec. Intertec products are sophisticated data processing units and are not sold or distributed for personal, family or household purposes.

This Class A equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instructions manual, may cause interference to radio communications. As temporarily permitted by regulation it has not been tested for compliance with the limits for Class A computing devices pursuant to Subpart I of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference. Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

(f) Software, including either source code, object code, or any computer program used in connection with our equipment, whether purchased directly from Intertec or from an independent source.

WAIVER OF ALL EXPRESS OR IMPLIED WARRANTIES

Our limited warranty to repair or replace defective parts or components for ninety (90) days after date of shipment from our Columbia plant is being offered in lieu of all express or implied warranties.

INTERTEC MAKES NO EXPRESS WARRANTY OTHER THAN THE LIMITED WARRANTY SET FORTH ABOVE, CONCERNING THIS PRODUCT OR ITS COMPONENTS, NOR DO WE IMPLIEDLY WARRANT ITS MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

All statements, technical information and recommendations contained in this document and related documents are based on tests we believe to be reliable, but the accuracy or completeness thereof is not guaranteed.

THE FOREGOING LIMITED WARRANTIES ARE IN LIEU OF ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, EXCEPT AS TO CONSUMER GOODS IN WHICH CASE THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY ONLY FOR THE PERIOD OF THE LIMITED WARRANTY.

PURCHASERS OF CONSUMER PRODUCTS SHOULD NOTE THAT SOME STATES DO NOT ALLOW FOR THE EXCLUSION OF CONSEQUENTIAL DAMAGES OR THE LIMITATION OR THE DURATION OF IMPLIED WARRANTIES SO THE ABOVE EXCLUSION AND LIMITATION MAY NOT BE APPLICABLE.

THIS LIMITED WARRANTY GIVES THE PURCHASER SPECIFIC LEGAL RIGHTS, AND THE PURCHASER MAY ALSO HAVE OTHER RIGHTS WHICH MAY VARY FROM STATE TO STATE.

LIMITATION OF REMEDIES

INTERTEC SHALL NOT BE LIABLE FOR ANY INJURY, LOSS OR DAMAGE, DIRECT OR CONSEQUENTIAL, TO PERSONS OR PROPERTY CAUSED EITHER DIRECTLY OR INDIRECTLY BY THE USE OR INABILITY TO USE ITS PRODUCT AND/OR DOCUMENTS. SUCH LIMITATION IN LIABILITY SHALL REMAIN IN FULL FORCE AND EFFECT EVEN WHEN INTERTEC MAY HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH INJURIES, LOSSES OR DAMAGES.

Before purchasing or using, the Customer shall determine the suitability of Intertec's products and documents for his intended use and assumes all risk and liability whatsoever in connection therewith.

THE LIMITED WARRANTY TO REPLACE OR REPAIR PARTS OR COMPONENTS FOR (90) DAYS IS THE EXCLUSIVE REMEDY PROVIDED TO THE CUSTOMER AND THE LIABILITY OF INTERTEC WITH RESPECT TO ANY OTHER CONTRACT, SALE OR ANYTHING DONE IN CONNECTION THEREWITH, WHETHER IN CONTRACT, IN TORT, UNDER ANY WARRANTY, OR OTHERWISE, SHALL NOT EXCEED THE PRICE OF THE PART OR COMPONENT ON WHICH SUCH LIABILITY IS BASED.

Rights under this warranty are not assignable without the express prior consent, in writing, of Intertec Data Systems Corporation and, regarding the terms of such consent in writing, such assignee shall have no greater rights than his assignor.

In the event the Customer has any problem or complaints arising out of any breach of our limited warranty, including a failure to make repairs in accordance with this warranty, or unsuccessful repair attempts by an authorized repair facility, the Customer is encouraged to inform Intertec, in writing, of his or her problem or complaint. Any such writing should be addressed to Intertec Data Systems Corporation at 2300 Broad River Road, Columbia, South Carolina 29210, and should be marked with the phrase "Warranty Claim."

CORPORATE HEADQUARTERS • 2300 BROAD RIVER ROAD • COLUMBIA, SOUTH CAROLINA 29210 • 803 798-9100 • TWX: 810-666-2115